THÈSE DE DOCTORAT

ANALYSE DE TRAFIC ROUTIER À PARTIR DE VIDÉOS À FAIBLE DÉBIT

Par

Zhiming Luo

Cotutelle de Thèse

Présentée au Département d'informatique

En vue de l'obtention du grade de Philosophiae Doctor (Ph.D.)

FACULTÉ DES SCIENCES, UNIVERSITÉ DE SHERBROOKE

Présentée à Département des sciences cognitives

En vue de l'obtention du grade de Philosophiae Doctor (Ph.D.)

UNIVERSITÉ DE XIAMEN

Sherbrooke, Québec, Canada, Janvier 2018

DOCTORAL THESIS

TRAFFIC ANALYSIS OF LOW AND ULTRA-LOW FRAME-RATE VIDEOS

By

Zhiming Luo

Co-advised thesis

Presented at the Department of Computer Science

For the degree of Doctor of Philosophy (Ph.D.)

FACULTY OF SCIENCE, UNIVERSITY OF SHERBROOKE

Presented at the Department of Cognitive Science

For the degree of Doctor of Philosophy (Ph.D.)

XIAMEN UNIVERSITY

Sherbrooke, Québec, Canada, January 2018

Le 31 Janvier 2018

Le jury a accepté le mémoire de Monsieur Zhiming Luo dans sa version finale.

Membres du jury

Professeur Pierre-Marc Jodoin
Directeur de recherche
Département d'informatique, Université de Sherbrooke

Professeur Shaozi Li
Directeur de recherche
Cognitive Science Department, Xiamen University

Professeur Songzhi Su
Membre interne
Cognitive Science Department, Xiamen University

Professeur Guorong Cai
Membre externe
School of Science, Jimei University

Professeur Jean Rouat
Prśident-rapporteur
Département de génie électrique et de génie informatique,
Université de Sherbrooke

Summary

Nowadays, traffic analysis are relying on data collected from various traffic sensors. Among the various traffic surveillance techniques, video surveillance systems are often used for monitoring and characterizing traffic load. In this thesis, we focused on two aspects of traffic analysis without using motion features in low frame-rate videos : Traffic density flow analysis and Vehicle detection and classification.

Traffic density flow analysis : Knowing in real time when the traffic is fluid or when it jams is a key information to help authorities re-route vehicles and reduce congestion. Accurate and timely traffic flow information is strongly needed by individual travelers, the business sectors and government agencies. In this part, we investigated the possibility of monitoring highway traffic based on videos whose frame rate is too low to accurately estimate motion features.

As we are focusing on analyzing traffic images and low frame-rate videos, traffic density is defined as the percentage of road being occupied by vehicles. In our previous work [99], we validated that traffic status is highly correlated to its texture features and that Convolutional Neural Networks (CNN) has the superiority of extracting discriminative texture features. We proposed several CNN models to segment traffic images into three different classes (road, car and background), classify traffic images into different categories (empty, fluid, heavy, jam) and predict traffic density without using any motion features. In order to generalize the model trained on a specific dataset to analyze new traffic scenes, we also proposed a novel transfer learning framework to do model adaptation.

Summary

Vehicle detection and classification : The detection of vehicles pictured by traffic cameras is often the very first step of video surveillance systems, such as vehicle counting, tracking and retrieval. In this part, we explore different deep learning methods applied to vehicle detection and classification.

Firstly, realizing the importance of large dataset for traffic analysis, we built and released the largest traffic dataset (MIO-TCD) in the world for vehicle localization and classification in collaboration with colleagues from Miovision inc. (Waterloo, On). With this dataset, we organized the Traffic Surveillance Workshop and Challenge in conjunction with CVPR 2017.

Secondly, we evaluated several state-of-the-art deep learning methods for the classification and localization task on the MIO-TCD dataset. In light of the results, we may conclude that state-of-the-art deep learning methods exhibit a capacity to localize and recognize vehicle from a single video frame. While with a deep analysis of the results, we also identify scenarios for which state-of-the-art methods are still failing and propose concrete ideas for future work.

Lastly, as saliency detection aims to highlight the most relevant objects in an image (e.g. vehicles in traffic scenes), we proposed a multi-resolution 4×5 grid CNN model for the salient object detection. The model enables near real-time high performance saliency detection. We also extend this model to do traffic analysis, experiment results show that our model can precisely segment foreground vehicles in traffic scenes.

Keywords: Traffic analysis ; traffic density ; video surveillance ; vehicle localization ; vehicle classification ; saliency detection ; deep learning ; convolutional neural networks.

Sommaire

De nos jours, l'analyse de trafic routier est de plus en plus automatisée et s'appuie sur des données issues de senseurs en tout genre. Parmi les approches d'analyse de trafic routier figurent les méthodes à base de vidéo. Les méthodes à base de vidéo ont pour but d'identifier et de reconnaître les objets en mouvement (généralement des voitures et des piétons) et de comprendre leur dynamique. Un des défis parmi les plus difficile à résoudre est d'analyser des séquences vidéo dont le nombre d'images par seconde est très faible. Ce type de situation est pourtant fréquent considérant qu'il est très difficile (voir impossible) de transmettre et de stocker sur un serveur un très grand nombre d'images issues de plusieurs caméras. Dans ce cas, les méthodes issues de l'état de l'art échouent car un faible nombre d'images par seconde ne permet pas d'extraire les caractéristiques vidéos utilisées par ces méthodes tels le flux optique, la détection de mouvement et le suivi de véhicules. Au cours de cette thèse, nous nous sommes concentré sur l'analyse de trafic routier à partir de séquences vidéo contenant un très faible nombre d'images par seconde. Plus particulièrement, nous nous sommes concentrés sur les problème d'estimation de la densité du trafic routier et de la classification de véhicules. Pour ce faire, nous avons proposé différents modèles à base de réseaux de neurones profonds (plus particulièrement des réseaux à convolution) ainsi que de nouvelles bases de données permettant d'entraîner les dits modèles. Parmi ces bases de données figure « MIO-TCD », la plus grosse base de données annotées au monde faite pour l'analyse de trafic routier.

Mots-clés: Analyse de trafic routier ; densité de trafic ; surveillance vidéo ; classification de véhicule, apprentissage profond, réseaux à convolution

Remerciements

Firstly, I must thank my two supervisors : Prof. Pierre-Marc Jodoin and Prof. Shaozi Li. I've been very fortunate to get the opportunity of studying in Canada with Prof. Jodoin. Under his supervision for the past three years, I have learned a lot about how to do research and gained lots of professional knowledge. Prof. Li always tell me to work hard, think deep about your project and teach me a lot of life experiences.

Besides my supervisors, I would like to thank Prof. Hugo Larochelle and Prof. Songzhi Su for their insights and various discussions on my projects.

I have to thank many people who made my life at Xiamen University so wonderful. Thank you all the members in the IMT lab for all the nice experiences. Thank you Yong Chen, Xiaodong Xie, Ling Luo and Rui Li for our amazing friendship. Thank you Wei Liu and Yuanzhen Cai for all the wonderful discussions. Thank you Zhun Zhong and Hong Liu for making our Hawaii CVPR week so remarkable.

For the past three years in Canada, I have enjoyed the beauty of colorful maple leaves and the wonderful white snow. There are many people I want to thank for their help of my stay in Canada. I would like to thank to my fellow VITAL lab-mates : Mohammad, Yi, Martin, Sebastian and Clement. Also I want to thank my former colleagues at the Miovision : Justin Eichel, Andrew Achkar and Akshaya Mishra.

In the end, I want to thank my family, who always supported and encouraged me through the years, your expectations have always been my forward momentum.

Table des matières

Table des figures

Liste des tableaux

Introduction

Overview

Nowadays, traffic analysis depends on data collected from various sources, including inductive loops, radars, cameras, mobile Global Positioning System, crowdsourcing, social media, etc. [12]. With the widespread use of various traffic sensors, the total amount of traffic data to be analyzed is exploding as we have entered into the era of big transportation data. Transportation management and control is now more data driven than ever.

Among the various traffic surveillance techniques, video surveillance systems are often used for monitoring and characterizing traffic load. As such there is an urgent need for intelligent transportation systems to allow real time monitoring of roads. In that perspective, computer vision has attracted a great deal of attention and made significant contributions to various applications such as vehicle counting, vehicle tracking, and behavior analysis [18, 128].

To our knowledge, current camera-based traffic monitoring systems all rely on motion features. These features are typically estimated with optical flow, motion detection and vehicle tracking [93, 149]. Motion features are used to count the number of vehicles on the road, estimate traffic speed, and recover global motion trajectories [135]. But these traffic monitoring systems share a fundamental limitation as they all need high frame rate videos to extract reliable motion features.

Unfortunately, low frame-rate videos are very common as many large-scale camera networks cannot stream and store high-frame rate videos gathered by hundreds of cameras. In a typical traffic surveillance system, the central subsystem connects all terminal cameras through a private network where the video feed is stored and

analyzed. In practice, however, only parts of video streams can be simultaneously downloaded due to bandwidth limitation [60]. As a result, cameras often send to the server one frame every 2 or 3 seconds. It is also the case for IP cameras transmitting over a WIFI network or via the cellular network. The limited bandwidth makes it hard (and sometimes expensive) to have more than 2 frames per second. In such cases, one can only analyze traffic based on low frame rate videos out of which almost no motion features can be extracted.

In this thesis, we focus on analyzing traffic activities with low frame rate videos. The whole work can be divided in two parts namely : Traffic density flow analysis and Vehicle detection and classification, both in the context of low frame-rate videos.

1. Traffic density flow analysis

Knowing in real time when the traffic is fluid of when it jams is a key information to help authorities re-route vehicles and reduce congestion. Accurate and timely traffic flow information is strongly needed by individual travelers, the business sectors and government agencies. It has the potential to help road users make better travel decisions, alleviate traffic congestion, reduce carbon emissions, and improve traffic fluidity.

As mentioned before, all traffic density estimation methods [20, 129, 32, 6] rely on motion features and thus are unfit to work on low or very-low framerate videos (less than 1 frame per second). One objective with this thesis is to analysis traffic density flow with low framerate videos without using motion features. For traffic images with different density flow (typically *empty, fluid, heavy* and *jam* as showed in Figure 1), we want to infer the traffic status by only processing 2D images and not videos. In order to do this, we formulated a basic assumption which is that traffic density is highly correlated to its 2D appearance. In other words, we assume that there is a correlation between the appearance of an image and the amount of vehicles on the road. In order to validate that hypothesis, we proposed several methods to recognize traffic activity without using any motion features. And in our previous work [99], we validated the hypothesis that the amount of traffic can be correlated to the content of a 2D image without the use of temporal information.

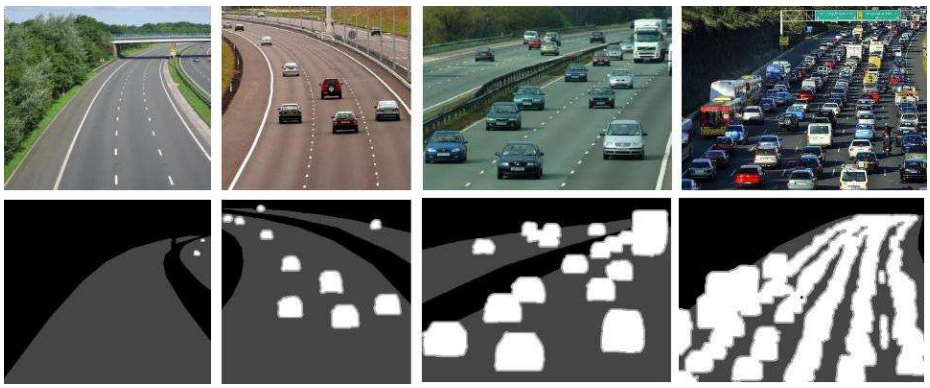

Figure 1 – Images from our Motorway Dataset show different traffic density. Each image has been manually segmented into 3 semantic classes (Road, Car, Background).

Convolutional neural networks (CNN) [82] are biological-inspired hierarchical multi-layer neural networks. Recently, methods using a CNN have established state-of-the-art performances in numerous computer vision tasks such as image classification [80, 127], object detection [54, 44], and action recognition [66, 126]. CNN has the superior ability of extracting discriminate texture features directly learned from the data. The first part of this thesis focuses on convolutional neural networks(CNN) for automatically localizing the road, the vehicles and the background and then estimate the overall traffic density.

One important problem that we had to face at the beginning of this thesis is the lack of a comprehensive and well-annotated dataset that we could rely on to train and test our models. As a consequence, we built our own dataset which we called the **Motorway Dataset**. This dataset contains 400 traffic images all taken from different highway cameras deployed in the UK. These images are divided into 4 categories based on their traffic density, namely *empty, fluid, heavy* and *jam*. Each category contains 100 images, and each image has been manually labeled into 3 different semantic classes *Road, Car*, and *Background*. Figure 1 contains some examples of the Motorway dataset. The Motorway dataset can be used to analyze traffic status by training a model on images taken by some cameras and test it on images taken by other cameras.

2. Vehicle detection and classification.

The detection of vehicles pictured by traffic cameras is often the very first step of video surveillance systems [18]. Such procedure is important to count cars, keep track of vehicles, retrieve specific vehicles, recognize different types of actions, and perform anomaly detection to name a few. The second objective of this thesis is to locate and classify vehicles. As shown in Figure 2, given a single 2D image (not a video) the goal is to put a bounding box around every vehicle and identify the vehicles sub-types.

Figure 2 – Sample images from the MIO-TCD localization dataset.

As we know, a plethora of deep learning methods have been recently published, most of which being surprisingly accurate at localizing and identifying various kinds of objects. This was made possible with the availability of large and well-annotated public datasets such as Pascal VOC [38], ImageNet [123] and Microsoft Coco [91]. Unfortunately, despite the number of publications devoted to traffic analysis, no such traffic dataset has been made available so far. The lack of such dataset has a number of implications, the most important one being that deep architectures trained on non-traffic oriented datasets such as ImageNet and Microsoft Coco do not generalize well to traffic images.

In order to answer that question, in collaboration with colleagues from Miovision inc. (Waterloo, On), we built and released the largest traffic dataset in the world. The dataset (which is called "MIOvision Traffic Camera Dataset **(MIO-TCD)**") contains 786,702 annotated images acquired at different times of the day and different

periods of the year by nearly a thousand traffic cameras deployed across Canada and the United States. The dataset consists of two parts : a "localization dataset", containing 137,743 full video frames with bounding boxes around traffic objects, and a "classification dataset", containing 648,959 crops of traffic objects from 11 classes. These images have been selected to cover a wide range of localization challenges and are representative of typical visual data captured today in urban traffic scenarios. Each moving object has been carefully outlined and identified by nearly 200 persons to enable a quantitative comparison and ranking of various algorithms.

Recognizing the importance of datasets to traffic analysis, we show how well-trained, state-of-the-art deep learning methods can be used to localize and identify moving objects with high precision without having to rely on motion features. Several state-of-the-art deep learning methods for the classification and localization task have been evaluated on the MIO-TCD dataset, and also we identify scenarios for which state-of-the-art methods are still failing and propose concrete ideas for future work. We believe that this work will have a substantial impact as the need for traffic monitoring systems working on still 2D images will keep increasing in the near future.

In traffic scenes, foreground vehicles on the road are the most important objects for understanding traffic activities, and the main goal of saliency detection is to highlight the most relevant objects in an image. Besides, due to the limited variety of objects pictured in traffic scenes, we explore the possibility of segmenting foreground vehicles based on a saliency detection model without using any motion information. As such, we proposed a simplified convolutional neural network which combines local and global information which can be applied both for saliency detection and vehicle segmentation. Our method can achieve start-of-the-art performance on several standard saliency benchmark datasets. We also tested the model for foreground vehicle segmentation, and it works surprisingly well on traffic images.

Outline and contributions

In this thesis, we developed different kinds of convolutional neural networks applied to traffic analytic.

In **Chapter 1**, we provide mathematical background knowledge for supervised

learning, optimization, SVM, neural network, and convolutional neural networks.

In **Chapter 2** we investigate the possibility of monitoring highway traffic based on videos whose frame rate is too low to accurately estimate motion features. We proposed several CNN models to segment traffic images into three different classes (road, car and background), classify traffic images into different categories (empty, fluid, heavy, jam) and predict traffic density without using any motion features. In order to generalize the model trained on a specific dataset to analyze new traffic scenes, we also proposed a novel transfer learning framework to do model adaptation. All the work in this chapter has been published as a journal paper in IEEE Transactions on Circuits and Systems for Video Technology.

> **Luo Z.**, Jodoin P.-M., Su S.-Z., Li S.-Z., Larochelle H., "*Traffic Analytics with Low Frame Rate Videos*", IEEE Transactions on Circuits and Systems for Video Technology, 2016.

In **Chapter 3**, we introduce a new benchmark dataset (MIO-TCD) for vehicle localization and classification, which is the largest fully annotated traffic surveillance dataset ever released to the public. With this dataset, we organized the Traffic Surveillance Workshop and Challenge in conjunction with CVPR 2017 and built an online evaluation system. We evaluated several state-of-the-art deep learning methods for the classification and localization task on the MIO-TCD dataset, and also identify scenarios for which state-of-the-art methods are still failing and propose concrete ideas for future work. This led to a journal survey paper submitted to IEEE Transactions on Image Processing :

> **Luo Z.**, B.-Charron F., Lemaire C., Konrad J., Li S., Mishra A., Achkar A., Eichel J., Jodoin P.-M., "*MIO-TCD : A new benchmark dataset for vehicle classification and localization*", submitted to IEEE Transactions on Image Processing, 2017

In **Chapter 4**, we address the issue of saliency detection which aims to highlight the most relevant objects in an image (eg. vehicles in traffic scenes). We propose a simplified convolutional neural network which combines local and global information through a multi-resolution 4×5 grid structure, and the model is trained based on loss function inspired by the Mumford-Shah (MS) functional [111] which penalizes error

on the boundary. The model enables near real-time high performance saliency detection. We also extend to use this model to do traffic analysis (vehicle segmentation), experiments on various traffic videos demonstrate our model can do accurate vehicle segmentation with wonderful generalization abilities. The saliency model proposed in this Chapter has been published in the most prestigious and selective conference in computer vision : 2017 IEEE Conference on Computer Vision and Pattern Recognition, (CVPR 2017).

> **Luo Z.**, Mishra A., Achkar A., Eichel J., Li S.-Z., Jodoin P.-M., "*Non-Local Deep Features for Salient Object Detection*", IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2017).

Finally, in **Chapter 5** we identify the remaining challenges and discuss future directions.

Chapter 1

Machine Learning Basics

Machine learning is the core application of artificial intelligence, which focuses on constructing data analysis models that can automatically improve with experience. Machine learning allows computers to find hidden insights that can later on be used for analyzing unseen data. Machine learning algorithms are usually categorized into three broad families:

Supervised Learning: learning algorithms are presented with a set of examples where the desired output targets (labels) are already known.

Unsupervised learning: Unlabeled data or data with unknown structures are given to the learning algorithms to discover the hidden patterns from the data.

Reinforcement Learning: A system interacts with a dynamic environment, and needs to find the best possible actions given the overall context, i.e. the actions that would maximize a reward.

Between supervised and unsupervised learning is **semi-supervised learning**. In this case, machine learning methods have to deal with incomplete training signal (a training set with some missing target outputs). For the purpose of this thesis, we only provide the necessary technical background on supervised learning, SVM and neural networks.

1.1 Supervised Learning

Given a set of n training samples of the form $D = \{(\boldsymbol{x}_1, y_1), ..., (\boldsymbol{x}_n, y_n)\}$, such that $\boldsymbol{x_i} \in R^d$ is the feature vector of the ith example and y_i is the target. If the target is a real value, then we will be talking about a regression problem whereas if the target is a finite set, we will be talking about a classification problem. A machine learning method seeks a mapping $f : X \to Y$, where X is the input space and Y is the output space. For instance, X could be the space of images and Y could be 0 or 1 indicating whether there is dog inside the image or not. Unfortunately, in many cases it is difficult to manually specify the function f (e.g. it is unclear how to write down a mathematical function that can recognize a dog). Supervised learning algorithms provide an approach to learn the mapping from X to Y directly from the data. In this subsection we will first introduce the overall objective of supervised learning, then talk about regularization terms which can be used to reduce the model complexity, and finally give a brief introduction on optimization methods.

1.1.1 Objective

We assume that there is a joint probability distribution $p(X, Y)$ over X and Y, and each sample $(\boldsymbol{x_i}, y_i)$ in training set D is drawn identically and independently from $p(X, Y)$. A loss function $L(\hat{y}, y)$ is used to measure the disagreement between a predicted label $\hat{y}_i = f(\boldsymbol{x}_i)$ of a hypothesis and the true label y_i. The $risk$ associated with the hypothesis $f(\boldsymbol{x})$ is then defined as the expectation of the loss function:

$$R(f) = \int L(f(\boldsymbol{x}), y)p(\boldsymbol{x}, y)d\boldsymbol{x}dy \tag{1.1}$$

The ultimate goal of machine learning is to find $f^* \in F$ that minimizes this risk:

$$f^* = \arg\min_{f \in F} R(f). \tag{1.2}$$

Since the distribution $P(\boldsymbol{x}, y)$ is usually unknown to the learning algorithm, it is impossible to compute the risk $R(f)$. A common alternative is to use the **empirical**

risk which is computed by averaging the loss function on the training set:

$$f^* \approx \arg\min_{f \in F} \frac{1}{n} \sum_{i}^{n} L(f(\boldsymbol{x_i}), y_i). \tag{1.3}$$

Then the goal of machine learning is to solve the above optimization problem.

Given a loss function L, we can evaluate the quality of the estimated model f by computing a **training error** and a **testing error**. The training error indicates how well the model fits on the training set, and the testing error means the generalization ability of the model to analyze unseen data (typically data from a "testing" set). Whenever a model has a high training error, we will say that the model is **underfitting** the data. If on the other hand the model has a low training error but a much larger testing error, we will say that the model has **overfitted** the training data. In practice, we want the algorithm to learn a model that has a low training error and a low testing error.

1.1.2 Regularization

The hypothesis space F can contain functions f with various complexity level (e.g. different number of parameters). For instance, we could get a very complex high-order function $\hat{f}$ with zero training error for Equation (1.3), but with high testing error for samples not in the training set. As such, we can not expect this function $\hat{f}$ to generalize well on data $(\boldsymbol{x_i}, y_i)$ that the system has not seen during training. Another concern is that there may be different functions that can achieve the same training loss L, but with different generalization losses. In practice, we want the function f to depict the real data distribution with appropriate parameters. In that perspective, a regularization term R is often added to the objective:

$$f^* \approx \arg\min_{f \in F} \frac{1}{n} \sum L(f(\boldsymbol{x}_i), y_i) + R(f). \tag{1.4}$$

The goal of the regularization term $R(f)$ is to penalize complex functions f. A function f that minimizes Equation (1.4) is one which fits well on the data (i.e. minimizes the loss L) while being as simple as possible (i.e. small R). The most

widely used regularization terms are L2 and L1 norms as showed below:

$$\text{L2 norm: } R(f) = \lambda \|\boldsymbol{w}\|_2 \tag{1.5}$$

$$\text{L1 norm: } R(f) = \lambda \|\boldsymbol{w}\|_1 \tag{1.6}$$

where $\boldsymbol{w}$ is the to-be-learned parameter vector of the function f.

1.1.3 Optimization

The task of learning a model can also be formulated as an optimization problem of the form $\boldsymbol{w}^* = \arg\min_{\boldsymbol{w}} g(\boldsymbol{w})$, where $\boldsymbol{w}$ is the parameter vector of a model and g is the loss function which usually combines the average empirical risk $\sum_i L(f(\boldsymbol{x_i}), y_i)$ and the regularization term $R(f)$.

There are different methods for solving this optimization problem, such as *gradient descent*, *genetic algorithms*, *hill climbing*, etc. Among all these techniques, *gradient descent* is the most widely used. The *gradient descent* consists of two steps: (1) compute the gradient of the parameters, (2) update the parameters by taking a small step in the direction of the negative gradient.

Stochastic Gradient Descent (SGD): Since the dataset used for training a model can be very large (e.g. ImageNet has 1 million training images), it is difficult to compute the exact gradient over the whole training set, mainly for memory issues. In practice, we only sample a small mini-batch of examples to estimate the gradient. This strategy works pretty well in most practical applications. This algorithm is called Stochastic Gradient Descent (SGD) and is summarized in Algorithm 1.

Algorithm 1 Stochastic Gradient Descent

Initialize the starting point for $\boldsymbol{w}$.

repeat

1. Sample a mini-batch of m examples $\{(\boldsymbol{x}_1, y_1), ..., (\boldsymbol{x}_m, y_m)\}$ from the training set D.
2. Estimate the gradient $\nabla_{\boldsymbol{w}} g(\boldsymbol{w}) \approx \nabla_{\boldsymbol{w}} [\frac{1}{m} \sum_{i=1}^{m} L(f(\boldsymbol{x_i}), y_i) + R(f)]$
3. Compute the update direction: $\Delta \boldsymbol{w} = -\lambda \nabla_{\boldsymbol{w}} g(\boldsymbol{w})$
4. Update the parameters: $\boldsymbol{w} \leftarrow \boldsymbol{w} + \Delta \boldsymbol{w}$

until convergence

A critical hyper-parameter in SGD is the learning rate λ. If the learning rate is set too high, the whole optimization may not converge or even diverge. If it is set too low, the learning process will take too long before to converge. A common practice is to start from a high initial learning rate and then reduce it by some factor after every few iterations.

Momentum: In practice, one can obtain faster converge speed by modifying the computation step of the update direction (Step 3 in Algorithm 1). *Momentum* is a technique inspired by the basic physics of moving particles. The update scheme of *Momentum* is designed to encourage the update to accelerate along a small and consistent directions of the gradient. The whole scheme can be summarized as follows:

$$\boldsymbol{v_t} \leftarrow \alpha \boldsymbol{v_{t-1}} - \lambda \nabla_{\boldsymbol{w}} g(\boldsymbol{w}) \tag{1.7}$$

$$\boldsymbol{w_t} \leftarrow \boldsymbol{w_{t-1}} + \boldsymbol{v_t} \tag{1.8}$$

where α is a hyper-parameter controlling the influence of previous updating directions that we usually set to 0.9, and $\boldsymbol{v_t}$ was initialized to be 0. The weight vector $\boldsymbol{w}$ can be thought of as a particle traveling through parameter space, and the negative gradient of the loss is used to accelerate the velocity of that particle. Unlike classical stochastic gradient descent, $\boldsymbol{w}_t$ tends to keep traveling in the same direction, preventing oscillations, which in turn has an effect on the position of the particle.

There are also some more advanced optimization methods such as Adagrad [36], RMSprop [56], Adam [76] which can dynamically adjust the learning rate based on the running information of gradients.

Adagrad is an adaptive learning rate method which keeps track of the sum of squared gradients $\boldsymbol{v}_t$. It is then used to normalize the gradient at the update step:

$$\begin{aligned} \boldsymbol{v}_t &= \boldsymbol{v}_{t-1} + \boldsymbol{g}_t^2 \\ \boldsymbol{w_t} &\leftarrow \boldsymbol{w_{t-1}} - \lambda \frac{\boldsymbol{g}_t}{\sqrt{\boldsymbol{v}_t} + \epsilon} \end{aligned} \tag{1.9}$$

where $\boldsymbol{g}_t$ is the gradient $\nabla_{\boldsymbol{w}} g(\boldsymbol{w})$ at step t and ϵ is a small value to avoid division by zero which is usually set to $1e-8$. Notice that the weights that got high gradients will have their learning rate reduced rapidly, while weights with small gradients or

infrequent updates will have their learning rate reduced slowly.

RMSprop is a slight modification of Adagrad which uses a running average of squared gradients which can avoid the monotonically decreasing learning rate issue in Adagrad

$$\begin{aligned} \boldsymbol{v}_t &= \beta \boldsymbol{v}_{t-1} + (1-\beta)\boldsymbol{g}_t^2 \\ \boldsymbol{w_t} &\leftarrow \boldsymbol{w_{t-1}} - \lambda \frac{\boldsymbol{g}_t}{\sqrt{\boldsymbol{v}_t} + \epsilon} \end{aligned} \tag{1.10}$$

where β is a decay rate which is usually set to 0.9.

Adam is a recently proposed optimizer which looks like RMSprop with momentum. In addition to keep track of the running average of squared gradients $\boldsymbol{v}_t$, Adam also keeps a running average of past gradient $\boldsymbol{m}_t$, similar to momentum:

$$\begin{aligned} \boldsymbol{m}_t &= \beta_1 \boldsymbol{m}_{t-1} + (1-\beta_1)\boldsymbol{g}_t \\ \boldsymbol{v}_t &= \beta_2 \boldsymbol{v}_{t-1} + (1-\beta_2)\boldsymbol{g}_t^2. \end{aligned} \tag{1.11}$$

Since $\boldsymbol{m}_t$ and $\boldsymbol{v}_t$ are initialized to 0's, the authors of Adam observe that they are biased towards zero, especially during the initial steps. They deal with these biases by computing bias-corrected $\hat{\boldsymbol{m}}_t$ and $\hat{\boldsymbol{v}}_t$:

$$\begin{aligned} \hat{\boldsymbol{m}}_t &= \frac{\boldsymbol{m}_t}{1-\beta_1^t} \\ \hat{\boldsymbol{v}}_t &= \frac{\boldsymbol{v}_t}{1-\beta_2^t} \end{aligned} \tag{1.12}$$

After that, Adam uses the same rule than in RMSprop to update the weights:

$$\boldsymbol{w_t} \leftarrow \boldsymbol{w_{t-1}} - \lambda \frac{\hat{\boldsymbol{m}}_t}{\sqrt{\hat{\boldsymbol{v}}_t} + \epsilon} \tag{1.13}$$

where the default values for β_1 is 0.9 and β_2 is 0.999.

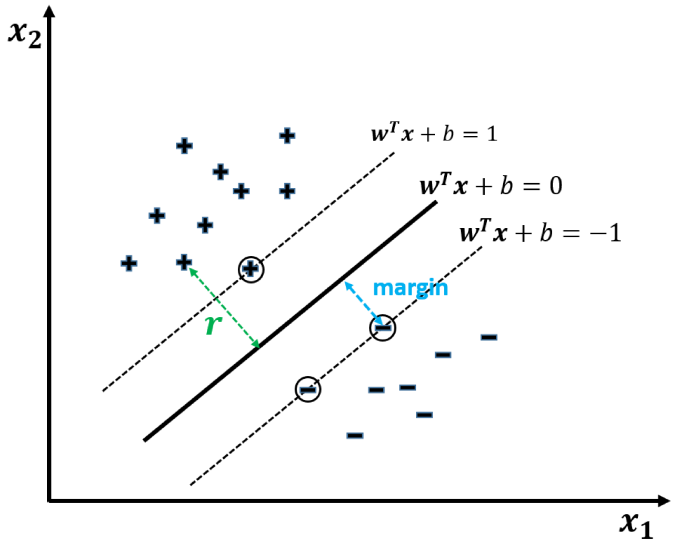

Figure 1.1 – The support vector (with a circle around the + and −) and the margin of a hyperplane $(\boldsymbol{w}, b)$.

1.2 SVM

The support vector machine (SVM) [27] is a linear binary classifier. Given a training set $D = \{(\boldsymbol{x_1}, y_1), ..., (\boldsymbol{x_n}, y_n)\}$, $y_i \in \{-1, 1\}$, SVM tries to find a hyperplane

$$\boldsymbol{w}^T \boldsymbol{x} + b = 0 \tag{1.14}$$

which maximizes the margin between the two classes. The overarching objective of this hyperplane is to correctly classify all training samples such that:

$$\begin{cases} \boldsymbol{w}^T \boldsymbol{x}_i + b > 0, & \text{if } y_i = +1 \\ \boldsymbol{w}^T \boldsymbol{x}_i + b < 0, & \text{if } y_i = -1. \end{cases} \tag{1.15}$$

As shown in Figure 1.1, the distance of a sample $\boldsymbol{x_i}$ to the hyperplane can be computed by

$$r_i = \frac{|\boldsymbol{w}^T \boldsymbol{x}_i + b|}{\|\boldsymbol{w}\|} = \frac{y_i(\boldsymbol{w}^T \boldsymbol{x}_i + b)}{\|\boldsymbol{w}\|}. \tag{1.16}$$

Proof. The distance of a point $\boldsymbol{x_i}$ to the hyperplane is calculated along a line parallel

to the unit normal vector $\boldsymbol{n}$ of the hyperplane $\boldsymbol{n} = \frac{\boldsymbol{w}}{\|\boldsymbol{w}\|}$. Let $\boldsymbol{x_0}$ be a point on the hyperplane which has $\boldsymbol{w}^T\boldsymbol{x_0} + b = 0$ and $\boldsymbol{v} = \boldsymbol{x_i} - \boldsymbol{x_0}$ is the vector from $\boldsymbol{x_0}$ to $\boldsymbol{x_i}$. The distance r of $\boldsymbol{x_i}$ to the hyperplane is simply the length of the projection of $\boldsymbol{v}$ onto the unit normal vector $\boldsymbol{n}$. Since $\boldsymbol{n}$ is unit length, this distance is the absolute value of the dot product $\boldsymbol{v} \cdot \boldsymbol{n}$,

$$r_i = |\boldsymbol{v} \cdot \boldsymbol{n}| = \frac{|\boldsymbol{w}^T(\boldsymbol{x_i} - \boldsymbol{x_0})|}{\|\boldsymbol{w}\|} = \frac{|\boldsymbol{w}^T\boldsymbol{x_i} + b|}{\|\boldsymbol{w}\|}. \tag{1.17}$$

As shown in Equation (1.15), if $y_i = 1$, then $\boldsymbol{w}^T(\boldsymbol{x_i} - \boldsymbol{x_0}) > 0$ and if $y_i = -1$, then $\boldsymbol{w}^T(\boldsymbol{x_i} - \boldsymbol{x_0}) < 0$, and we can have

$$r_i = \frac{|\boldsymbol{w}^T\boldsymbol{x_i} + b|}{\|\boldsymbol{w}\|} = \frac{y_i(\boldsymbol{w}^T\boldsymbol{x_i} + b)}{\|\boldsymbol{w}\|}. \tag{1.18}$$

□

Support vectors are those samples which are the closest to the hyperplane, i.e. those for which their distance $\frac{y(\boldsymbol{w}^T\boldsymbol{x}+b)}{\|\boldsymbol{w}\|}$ is the smalles. The **margin** is the perpendicular distance of support vectors to the hyperplane. The goal of SVM is to optimize the parameters $\boldsymbol{w}$ and b such that the data are correctly classified (Equation (1.15)) while maximizing the margin. This can be done by solving

$$\arg\max_{\boldsymbol{w},b} \left\{ \min_i \frac{y(\boldsymbol{w}^T\boldsymbol{x_i} + b)}{\|\boldsymbol{w}\|} \right\}. \tag{1.19}$$

It is very difficult to directly solve this optimization problem. Thus, we shall convert it into an equivalent problem which is easier to solve. Note that if we rescale $\boldsymbol{w} \to \alpha\boldsymbol{w}$ and $b \to \alpha b$, the distance from any sample $\boldsymbol{x_i}$ to the hyperplane $\frac{y_i(\boldsymbol{w}^T\boldsymbol{x_i}+b)}{\|\boldsymbol{w}\|}$ will remain unchanged [11]. We can use this freedom to set

$$y_i(\boldsymbol{w}^T\boldsymbol{x_i} + b) = 1 \tag{1.20}$$

for the support vectors. In this case, a set of parameters $(\boldsymbol{w}, b)$ that satisfies Equation (1.15) while having maximum margin implies that all samples $(\boldsymbol{x}_i, y_i)$ shall satisfy

the constrain:

$$y_i(\boldsymbol{w}^T\boldsymbol{x_i} + b) \geq 1. \tag{1.21}$$

According to this equation, is appears that $\min_i y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b) = 1$ and that Equation (1.19) can be rewritten as follows:

$$\begin{aligned} &\arg\max_{w,b} \frac{1}{\|\boldsymbol{w}\|} \\ &\text{s.t.} \quad y_i(\boldsymbol{w}^T\boldsymbol{x_i} + b) \geq 1, \quad i = 1, 2, ..., n. \end{aligned} \tag{1.22}$$

Since maximizing $\|\boldsymbol{w}\|^{-1}$ is equivalent to minimize the $\|\boldsymbol{w}\|^2$, we can rewrite the optimization criteria of Equation (1.22) as:

$$\begin{aligned} &\arg\min_{w,b} \frac{1}{2}\|\boldsymbol{w}\|^2 \\ &\text{s.t.} \quad y_i(\boldsymbol{w}^T\boldsymbol{x_i} + b) \geq 1, \quad i = 1, 2, ..., n. \end{aligned} \tag{1.23}$$

Equation (1.23) assumes that the training data is linearly separable, which means that it does not allow for any outlier. To improve the robustness of the model, a slack variable $\xi_i \geq 0$ is often used to relax the linearly separable constrain by allowing for wrong classifications. This soft-margin criteria modifies the constrain of Equation (1.23) to:

$$y_i(\boldsymbol{w}^T\boldsymbol{x_i} + b) \geq 1 - \xi_i. \tag{1.24}$$

If $0 < \xi_i < 1$, $\boldsymbol{x_i}$ violates the margin but is still classified correctly. However, if $\xi_i > 1$, $\boldsymbol{x_i}$ would be a wrong classification.

Since this soft-margin criteria allows $\boldsymbol{x_i}$ to violate the original constrain, another penalty term for each slack variable ξ_i is added to the original optimization equation which then becomes:

$$\begin{aligned} &\arg\min_{w,b,\xi_i} \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{n}\xi_i \\ &\text{s.t.} \quad y_i(\boldsymbol{w}^T\boldsymbol{x_i} + b) \geq 1 - \xi_i \\ &\qquad \xi_i \geq 0, \quad i = 1, 2, ..., n. \end{aligned} \tag{1.25}$$

where C is a hyper-parameter which balances the trade-of between the correctness

and robustness. A large C means the model will penalize more wrongly classified samples while small C makes the new constrains to be easily ignored.

The optimization problem of Equation (1.25) is referred to as the *primal problem* and can be solved by convex quadratic programming. Alternatively, another approach for minimizing Equation (1.25) is through the dual optimization. Based on the method of Lagrange multipliers, we can get the Lagrange expression of Equation (1.25) as:

$$L(\boldsymbol{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\mu}) = \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{n}\xi_i + \sum_{i}^{n}\alpha_i(1 - \xi_i - y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b)) - \sum_{i}^{n}\mu_i\xi_i. \quad (1.26)$$

where $\boldsymbol{\alpha}$ and $\boldsymbol{\mu}$ are known as Lagrange multipliers.

Since we want an optimal $(\boldsymbol{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\mu})$ which minimizes the loss function in Equation (1.25), we can assume that there exist a solution $(\boldsymbol{w}, b, \boldsymbol{\xi})$ with their partial derivatives of L equal to zero [11].

$$\frac{\partial L}{\partial \boldsymbol{w}} = \boldsymbol{w} + \sum_{i=1}^{n}\alpha_i(-y_i)x_i = 0 \quad \Rightarrow \quad \boldsymbol{w} = \sum_{i=1}^{n}\alpha_i y_i x_i \quad (1.27)$$

$$\frac{\partial L}{\partial \boldsymbol{b}} = \sum_{i=1}^{n}\alpha_i(-y_i) = 0 \quad \Rightarrow \quad \sum_{i=1}^{n}\alpha_i y_i = 0 \quad (1.28)$$

$$\frac{\partial L}{\partial \xi_i} = C - \alpha_i - \mu_i = 0 \quad \Rightarrow \quad \mu_i = C - \alpha_i. \quad (1.29)$$

Note that $\|\boldsymbol{w}\|^2 = \boldsymbol{w}^T\boldsymbol{w}$. If we substitute $\boldsymbol{w} = \sum_{i=1}^{n}\alpha_i y_i x_i$ and $\mu_i = C - \alpha_i$ to Equation (1.26), and through some mathematical derivations we get

$$\begin{aligned} L = \quad & \frac{1}{2}\sum_{i=1}^{n}\alpha_i y_i \boldsymbol{x_i}^T \sum_{j=1}^{n}\alpha_j y_j \boldsymbol{x_j} + C\sum_{i=1}^{n}\xi_i \\ & + \sum_{i}^{n}\alpha_i\left(1 - \xi_i - y_i(\sum_{j=1}^{n}\alpha_j y_j \boldsymbol{x_j}^T\boldsymbol{x}_i + b)\right) - \sum_{i}^{n}(C - \alpha_i)\xi_i \\ = \quad & -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j \boldsymbol{x_i}^T\boldsymbol{x_j} + \sum_{i=1}^{n}\alpha_i \end{aligned} \quad (1.30)$$

This new objective function is in term of α_i only and not $\boldsymbol{w}$ as before. Then we can get a new optimization equation known as the *dual problem*:

$$\begin{aligned} &\arg\max_{\boldsymbol{\alpha}} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \boldsymbol{x}_i^T \boldsymbol{x}_j \\ &\text{s.t.} \quad \sum_{i=1}^{n} \alpha_i y_i = 0, \\ &\qquad 0 \leq \alpha_i \leq C, \quad i = 1, 2, ..., n. \end{aligned} \tag{1.31}$$

where C is the regularization parameter which bounds the possible range of the α_i. However, a lot of α_i will be very close to zero in the final solution, and those samples $\boldsymbol{x}_i$ with non-zero α_i will be the *support vectors*.

Linear SVM is very efficient when most of the data is linearly separable. However, there are cases for which the data is non-linearly separable. For these cases, one can try to project the data $\boldsymbol{x}$ into another feature space $\phi(\boldsymbol{x})$ where data would be linarly separable. By doing so, Equation (1.31) can be re-formulated as:

$$\begin{aligned} &\arg\max_{\boldsymbol{\alpha}} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \phi(\boldsymbol{x}_i)^T \phi(\boldsymbol{x}_j) \\ &\text{s.t.} \quad \sum_{i=1}^{n} \alpha_i y_i = 0, \\ &\qquad 0 \leq \xi_i \leq C, \quad i = 1, 2, ..., n. \end{aligned} \tag{1.32}$$

Since $\phi(\boldsymbol{x}_i)^T \phi(\boldsymbol{x}_j)$ appears in pairs, we can replace the dot product between two feature vectors by a kernel function:

$$K(x_i, x_j) = \phi(\boldsymbol{x}_i)^T \phi(\boldsymbol{x}_j). \tag{1.33}$$

This is known as the kernel trick which is used in the dual problem formulation. An advantage of the kernel trick is extending the original linear SVM to learn non-linear classifiers.

1.3 Artificial Neural Networks

Artificial neural networks are computing systems used on a variety of tasks such as computer vision, speech recognition, machine translation, etc.

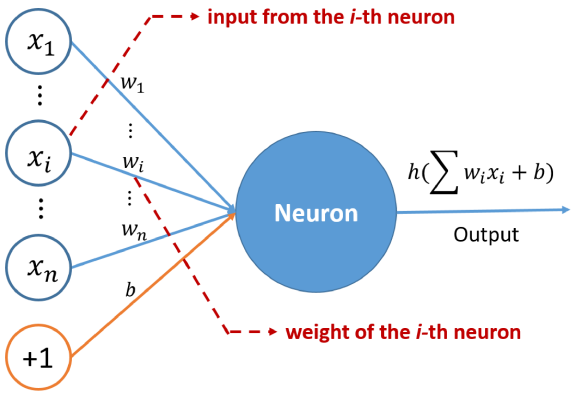

Figure 1.2 – A neuron computes the dot products between the input $\boldsymbol{x}$ and the weights $\boldsymbol{w}$ and adds a bias term b, then passes the result to an activation function h.

1.3.1 Perceptron

The Perceptron is the first neural network ever published [122]. The Perceptron is a binary linear classifier which learns a function that maps an input vector $\boldsymbol{x_i}$ to a binary output $f(\boldsymbol{x_i})$:

$$f(\boldsymbol{x}_i) = \begin{cases} 1 & \text{if } \boldsymbol{w}^T \boldsymbol{x_i} + b > 0 \\ -1 & \text{otherwise} \end{cases} \tag{1.34}$$

where $\boldsymbol{w}$ is the weights and b is the bias.

The **neuron** shown in Figure 1.2 is the basic computation block of artificial neural networks. A neuron computes the weighted sum of the input $\boldsymbol{x}$ with the weights $\boldsymbol{w}$, and then passes the result to an **activation function** h

$$f(\boldsymbol{x}) = h(\sum_i w_i x_i + b). \tag{1.35}$$

In the case of the Perceptron, the activation function is a simple sign function returning 1 for positive values and -1 otherwise. Notice that b can be treated as a dummy node with a constant input value 1 and a weight equals to b. As such, the original input vector for a sample $\boldsymbol{x_i}$ can be extended as $\hat{\boldsymbol{x}}_{\boldsymbol{i}} = [1, \boldsymbol{x}_{i,1}, ..., \boldsymbol{x}_{i,n}]^T$, and the weights will become $\hat{\boldsymbol{w}} = [b, w_1, ..., w_n]^T$.

For a given training set $D = \{(\hat{\boldsymbol{x}}_{\boldsymbol{1}}, y_1), ..., (\hat{\boldsymbol{x}}_{\boldsymbol{n}}, y_n)\}$, the Perceptron loss is defined as the sum over all wrongly classified training examples [11]:

$$L(\hat{\boldsymbol{w}}) = -\sum_{\hat{\boldsymbol{x}}_i \in M} y_i \hat{\boldsymbol{w}}^T \hat{\boldsymbol{x}}_{\boldsymbol{i}}, \tag{1.36}$$

where $y_i \in \{+1, -1\}$ is the desired output for sample $\hat{\boldsymbol{x}}_i$ and M is the set of all wrongly classified examples. L is a linear function with respect to $\hat{\boldsymbol{w}}$ and equals to zero when all examples are correctly classified. The gradient $\frac{\partial L}{\partial \hat{w}}$ can be computed as follows:

$$\frac{\partial L}{\partial \hat{\boldsymbol{w}}} = -\sum_{\hat{x}_i \in M} y_i \hat{\boldsymbol{x}}_{\boldsymbol{i}}. \tag{1.37}$$

This allows us to minimize L by using the gradient descent algorithm discussed in Section 1.1.3. At each step, $\lambda \sum_{\hat{x}_i \in M} y_i \hat{\boldsymbol{x}}_{\boldsymbol{i}}$ is added to the weight vector $\hat{\boldsymbol{w}}$ to do an update. If the training set is linearly separable, the Perceptron algorithm is guaranteed to find a zero-loss solution. However, if the training data is not linearly separable, the optimization will never converge.

1.3.2 Logistic Regression

The activation function of the Perception is a sign function which does not provide a probabilistic output. Instead, we can use a logistic sigmoid function ($\sigma(x) = \frac{1}{1+e^{-x}}$) as activation function which squash the output in the range of $[0, 1]$. With a sigmoid activation function, we can interpret a neural network as a function which estimates the conditional probability [112] of having a class $y = 1$ given an input vector $\boldsymbol{x}$

$$P(y = 1|\boldsymbol{x}, \boldsymbol{w}) = \sigma(\boldsymbol{w}^T \boldsymbol{x}) = f_{\boldsymbol{w}}(\boldsymbol{x}) \tag{1.38}$$

and

$$P(y = 0|\boldsymbol{x}, \boldsymbol{w}) = 1 - p(y = 1|\boldsymbol{x}, \boldsymbol{w}) = 1 - f_{\boldsymbol{w}}(\boldsymbol{x}). \tag{1.39}$$

Note that this can be written more compactly as:

$$P(y|\boldsymbol{x}, \boldsymbol{w}) = f_{\boldsymbol{w}}(\boldsymbol{x})^y (1 - f_{\boldsymbol{w}}(\boldsymbol{x}))^{1-y}. \tag{1.40}$$

The logistic sigmoid function can be considered as computing the class posterior probabilities $P(y=1|\boldsymbol{x})$ by using the Bayes theorem:

$$\begin{aligned} P(y=1|\boldsymbol{x}) &= \frac{P(\boldsymbol{x}|y=1)P(y=1)}{P(\boldsymbol{x}|y=0)P(y=0)+P(\boldsymbol{x}|y=1)P(y=1)} \\ &= \frac{1}{1+e^{-a}} = \sigma(a) \end{aligned} \tag{1.41}$$

where $a = \ln \frac{P(\boldsymbol{x}|y=1)P(y=1)}{P(\boldsymbol{x}|y=0)P(y=0)}$.

The goal of the logistic regression is to find parameters $\boldsymbol{w}$ which maximizes the likelihood of observing D. Assuming that the n training examples are independent, we can write down the class posterior probabilities with respect to parameters $\boldsymbol{w}$ as a Bernoulli distribution

$$L(\boldsymbol{w}) = \prod_{i=1}^{n} P(y_i|\boldsymbol{x}_i, \boldsymbol{w}) = \prod_{i=1}^{n} f_{\boldsymbol{w}}(\boldsymbol{x_i})^{y_i}(1-f_{\boldsymbol{w}}(\boldsymbol{x_i}))^{1-y_i}, \tag{1.42}$$

which we look forward to maximizing. Maximizing Equation (1.42) is equivalent to minimize the negative log likelihood :

$$l(\boldsymbol{w}) = -\log L(\boldsymbol{w}) = -\sum_{i=1}^{n} (y_i \log f_{\boldsymbol{w}}(\boldsymbol{x_i}) + (1-y_i)\log(1-f_{\boldsymbol{w}}(\boldsymbol{x_i})) \tag{1.43}$$

which is known as the *cross-entropy* loss. Since $\nabla_{\boldsymbol{x}}\sigma(x) = (1-\sigma(x))\sigma(x)$, the gradient of the loss function $l(\boldsymbol{w})$ with respect to $\boldsymbol{w}$ is:

$$\nabla_{\boldsymbol{w}} l(\boldsymbol{w}) = \sum_{i=1}^{n} (f_{\boldsymbol{w}}(\boldsymbol{x_i}) - y_i)\boldsymbol{x_i}. \tag{1.44}$$

After getting these gradients, we can use the gradient descent algorithms presented in Section 1.1.3 to find the optimal parameters $\boldsymbol{w}^*$.

We can generalize the two-class logistic regression model to a multi-class model (i.e. $y \in \{1, 2, ..., K\}$). The posterior probabilities for $K > 2$ can be computed by

$$P(y=k|\boldsymbol{x}) = \frac{P(\boldsymbol{x}|y=k)p(y=k)}{\sum_j P(\boldsymbol{x}|y=j)p(y=j)} = \frac{e^{a_k}}{\sum_j e^{a_j}} \tag{1.45}$$

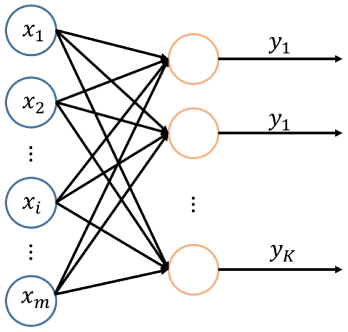

Figure 1.3 – A multiclass classification can be pictured as a two layers network with K output neurons (one for each class).

which is known as the *softmax function*. The quantities a_k are defined by

$$a_k = \ln P(\boldsymbol{x}|y = k)P(y = k). \tag{1.46}$$

If $a_k \gg a_j$ for all $j \neq k$, then $P(y = k|\boldsymbol{x}) \simeq 1$ and $P(y = j|\boldsymbol{x}) \simeq 0$. This is why the *softmax function* represents a smoothed version of the 'max' function.

As shown in Figure 1.3, a multiclass classification can be pictured as a two layers network with K output neurons (one for each class) [14]. We can get the posterior probabilities by softmax transformation of linear functions of the $\boldsymbol{x}$ as

$$P(y = k|\boldsymbol{x}, \boldsymbol{W}) = \frac{e^{\boldsymbol{w_k^T x_i}}}{\sum_{j=1} e^{\boldsymbol{w_j^T x}}} \tag{1.47}$$

where $\boldsymbol{W} = \{\boldsymbol{w}_1, \boldsymbol{w}_2, ..., \boldsymbol{w}_K\}$, and each $\boldsymbol{w}_j$ are the parameters for computing the 'activation' of each class.

Similar to the computation procedures of the negative log likelihood Equation (1.43), the loss function of multiclass logistic regression is computed as follows:

$$l(\boldsymbol{W}) = -\sum_{i=1}^{n}\sum_{j=1}^{K} \mathbf{1}\{y_i = k\} \log P(y = j|\boldsymbol{x}_i, \boldsymbol{W}) \tag{1.48}$$

where $\mathbf{1}\{\cdot\}$ is the indicator function returning 1 when $y_i == k$ and 0 otherwise. The

gradient of the loss function $l(\boldsymbol{W})$ with respect to each $\boldsymbol{w}_i$ is as follows:

$$\nabla_{\boldsymbol{w}_j} l(\boldsymbol{W}) = -\sum_{i=1}^{n} \Big[\boldsymbol{x}_i\Big(\mathbf{1}\{y_i = j\} - P(y = j|\boldsymbol{x}_i, \boldsymbol{W})\Big)\Big]. \tag{1.49}$$

Then the optimization methods discussed in Section 1.1.3 can be utilized to learn the parameters $\boldsymbol{W}$.

1.3.3 Multi-layer perceptron

As we know, the Perceptron and the logistic regression can only learn linear decision functions, while many problems involve non-linearly separable data. A multi-layer perceptron (MLP) is used to deal with these issues. The MLP is a class of artificial neural networks, which consists of at least three layers of neurons. The first layer is referred to as the input layer, the last layer is the output layer, and the other intermediate layers are known as hidden layers. The number of output neurons typically corresponds to the number of classes for a classification task. MLPs are often known as the "vanilla" neural networks. In Figure 1.4, we give an example of multi-layer perceptron with one input layer L_1, two hidden layers L_2 and L_3, and three output neurons in Layer L_4.

In order to add non-linearity into MLPs, each neuron in the hidden layers usually uses a non-linear **activation function**. In general, the activation function needs to be continuous and differentiable, and the followings are the most commonly-used activation functions:

Sigmoid: The sigmoid non-linearity has the mathematical form $\sigma(x) = \frac{1}{1+e^{-x}}$ which takes a real-valued number and "squashes" it into a range between 0 and 1.

Tanh: The tanh non-linearity computes the function $Tanh(x) = \frac{1-e^{-2x}}{1+e^{-2x}}$. The $Tanh$ function squashes a real-valued number within the range [-1, 1].

ReLU: The Rectified Linear Unit (ReLU) thresholds a number at zero by using the function $ReLU(x) = \max(0, x)$.

Leaky ReLU: Instead of thresholding to zero when $x < 0$, a leaky ReLU allows

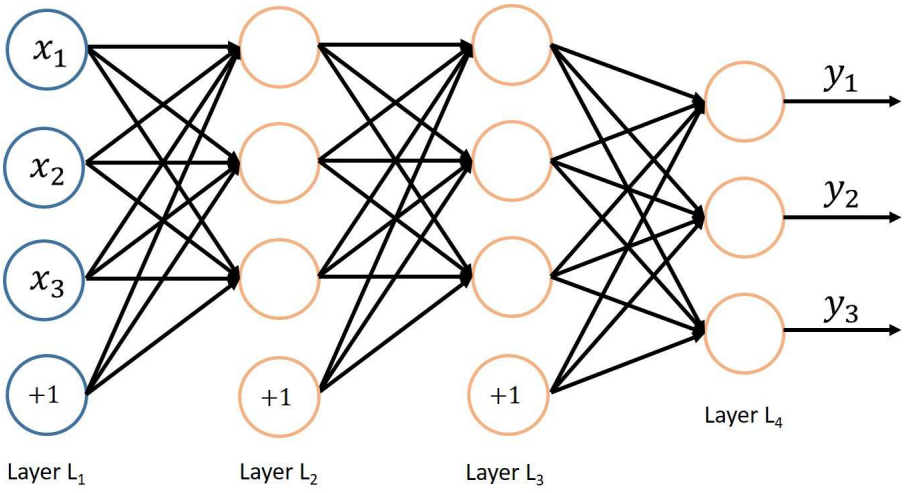

Figure 1.4 – A multi-layer perceptron with one input layer $L1$, two hidden layers $L2$ and $L3$, and three output neurons in Layer $L4$.

a small gradient when the unit is not active, as such it computes

$$h(x) = \begin{cases} x & \text{if } x >= 0 \\ \alpha x & \text{otherwise} \end{cases} \tag{1.50}$$

where α is a small constant (e.g. 0.1).

For a training set $D = \{(\boldsymbol{x}_1, \boldsymbol{y}_1), ..., (\boldsymbol{x}_n, \boldsymbol{y}_n)\}$, an MLP takes an $\boldsymbol{x}_i$ as input and gives a prediction $\hat{\boldsymbol{y}}_i$ (i.e. (y_1, y_2, y_3) in Figure 1.4). Then a loss function can be used to measure the discrepancy between the prediction $\hat{\boldsymbol{y}}_i$ and the ground truth $\boldsymbol{y}_i$. For example, a simple loss could be

$$L = \sum_{i}^{n} (\hat{\boldsymbol{y}}_i - \boldsymbol{y}_i)^2. \tag{1.51}$$

Like for the other methods we have seen before, the main goal when training an MLP is to find the parameters that minimize this loss function. In Section 1.1.3, we saw that if we can evaluate the gradients of the Loss with respect to the parameters $\boldsymbol{w}$, then a gradient descent optimizer can be used to minimize the loss function. As such, the main issue becomes how to compute the gradients of a MLP. In the next section,

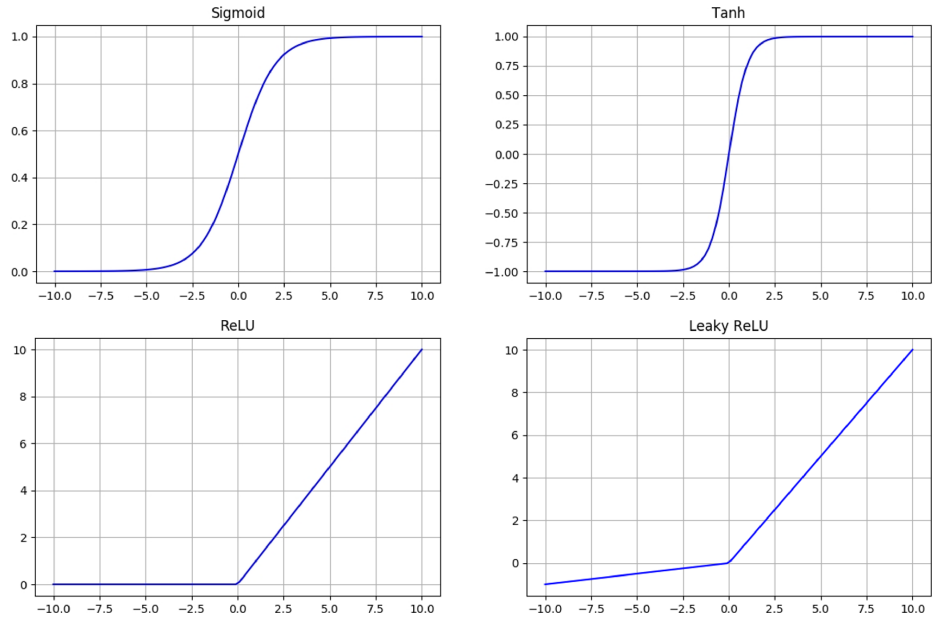

Figure 1.5 – The figure of the most commonly used activation functions.

we will discuss how to use the backpropagation algorithm to compute these gradients.

1.3.4 Backpropagation

The backpropagation algorithm is a method for computing gradients of the loss function with respect to each parameter w_i through a recursive application of the **chain rule**.

Chain rule: Let us start with a simple functions: $f(x, y) = (3x + 2)^2$ for which we want to compute $\frac{\partial f}{\partial x}$. Notice that this expression can be broken down into three intermediate functions: $a = 3x$, $b = a+2$ and $f = b^2$. The gradient of each individual function with respect to its input can be easily computed by: $\frac{\partial a}{\partial x} = 3$, $\frac{\partial b}{\partial a} = 1$ and $\frac{\partial f}{\partial b} = 2b$. Once we get all these local gradients, we can get the final gradient through a recursive backward pass $\frac{\partial f}{\partial x} = \frac{\partial f}{\partial b}\frac{\partial b}{\partial a}\frac{\partial a}{\partial x}$. The general formulation of the chain rule can be expressed as:

$$\text{if } y = f(u) \text{ and } u = g(x), \text{ then } \frac{\partial y}{\partial x} = \frac{\partial y}{\partial u}\frac{\partial u}{\partial x}. \tag{1.52}$$

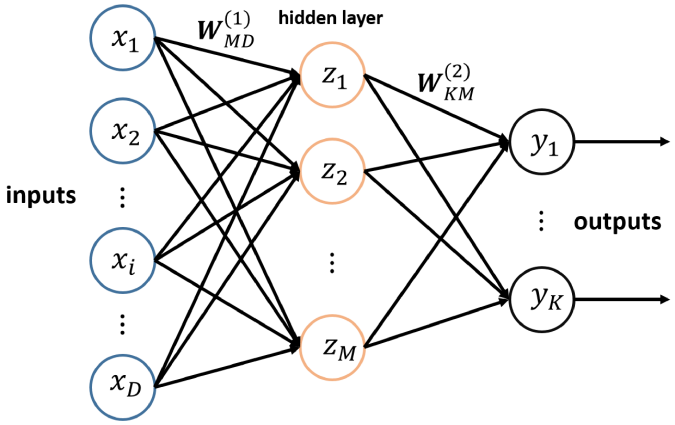

Figure 1.6 – A multilayer neural network which is corresponding to Equation (1.53). The input, hidden, and output variables are represented by nodes, and the weight parameters are represented by links between the nodes.

Notice that the chain rule is composed of several local procedures where local gradients of their inputs are respect to their output values. In the next paragraph, we discuss how to apply this chain rule to compute the gradient of a neural network.

As shown in Figure 1.6, we will demonstrate the backpropagation through a three layer MLP applied to a regression task with K outputs. In this case, each output y_k computes the following function:

$$y_k(\boldsymbol{x}, \boldsymbol{w}) = \sum_{m=1}^{M} w_{km}^{(2)} z_m = \sum_{m=1}^{M} w_{km}^{(2)} \sigma(\sum_{d=1}^{D} w_{md}^{(1)} x_d) \tag{1.53}$$

where $\sigma(\cdot)$ is a sigmoid activation function and $z_m = \sigma(\sum_{d=1}^{D} w_{md}^{(1)} x_d)$. Note that the superscript (1) and (2) indicate the layer number.

For this example, we will take the $L2$ loss to measure the disagreement between the outputs $\boldsymbol{y}$ and the real label $\boldsymbol{r}$ as:

$$L(\boldsymbol{y}, \boldsymbol{r}) = \frac{1}{2} \sum_{k=1}^{K} (y_k - r_k)^2. \tag{1.54}$$

In the following, we will show how to apply the chain rule to compute the gradients with respect to the parameters and neurons.

In the first step, the gradient of the loss with respect to each output can be computed as

$$\frac{\partial L}{\partial y_i} = y_i - r_i. \tag{1.55}$$

For the next step, since each parameter $w_{km}^{(2)}$ only connects the neuron z_m and y_k, then we can get the gradient of the loss with respect to the parameter $w_{km}^{(2)}$ as

$$\frac{\partial L}{\partial w_{km}^{(2)}} = \frac{\partial L}{\partial y_k}\frac{\partial y_k}{\partial w_{km}^{(2)}} = (y_k - r_k)z_m. \tag{1.56}$$

And as z_m connects to all the output neurons, we can get the gradient of the loss with respect to the neuron z_m as

$$\frac{\partial L}{\partial z_m} = \sum_{k=1}^{K}\frac{\partial L}{\partial y_k}\frac{\partial y_k}{\partial z_m} = \sum_{k=1}^{K}\frac{\partial L}{\partial y_k}w_{km}^{(2)}. \tag{1.57}$$

Finally, we can compute the gradient of the loss with respect to the parameters $w_{md}^{(1)}$ which connects input x_d and z_m as [11]

$$\frac{\partial L}{\partial w_{md}^{(1)}} = \frac{\partial L}{\partial z_m}\frac{\partial z_m}{\partial w_{md}^{(1)}} = \frac{\partial L}{\partial z_m}\frac{\partial \sigma(\sum_{d=1}^{D} w_{md}^{(1)}x_d)}{\partial z_m}. \tag{1.58}$$

As shown from the Equation (1.54) to Equation (1.58), for computing the gradient of the loss function with respect to each parameter $\frac{\partial L}{\partial w}$, one can simply compute the gradient through a reverse direction by applying the chain rule recursively from the output layer to the input layer, this process is also called as **backpropagation**. The backpropagation naturally extends to an arbitrary number of layers and to other derivable loss function. The backpropagation algorithm has been efficiently implemented using parallel vector operations and plays a fundamental role in the training of modern deep models.

1.3.5 Convolutional Neural Networks

Convolutional Neural Networks (CNNs, or ConvNet) [82] are neural networks architectures designed for processing data with some spatial or temporal topology

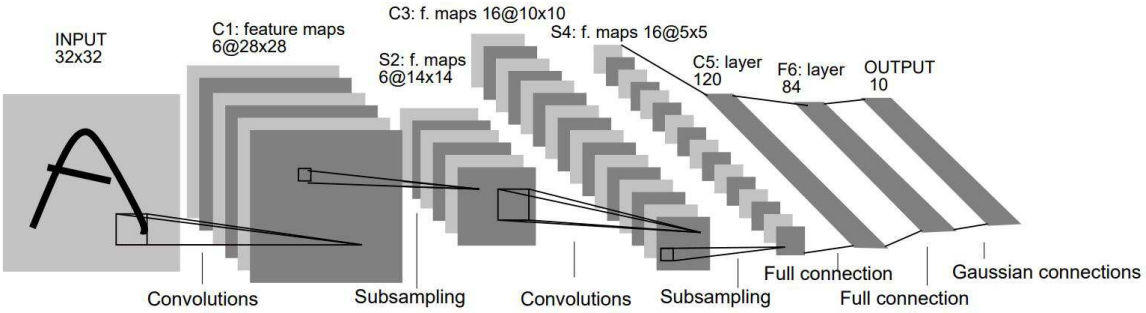

Figure 1.7 – The Architecture of LeNet-5, a Convolutional Neural Network used for digits recognition [82].

such as images and videos. The CNN has shown excellent performances in many computer vision and machine learning applications, such as image classification, image semantic segmentation, object detection, etc.

When processing images, a CNN usually takes an input tensor of order 3 (e.g. an image with height H, width W, and 3 channels ((R, G, B) color)). The input then sequentially goes through a series of processing steps to get the corresponding output. Such processing step in the context of a CNN is often called a layer. A layer could be a convolutional layer, an activation layer, a pooling layer, a fully-connected layer, etc. The main difference between various CNN models often comes from the number of layers they are made of and the way they are connected together. One of the representative architecture called "LeNet-5" is shown in Figure 1.7. This network was initially designed for handwritten isolated digit recognition.

Convolutional layer

A convolutional layer is the core computational block of a CNN. It is made of N filters which convert the input tensor (or features maps) X into N new feature maps. Convolution layers are typically organized in a cascaded fashion such that the output feature maps of one layer is the input of the subsequent one. Instead of connecting each element (or neuron) of a feature map to every output of previous layer as in the case for a MLP, the neurons of a convolution layer are connected to a limited numbers

1. Image from: https://cs.nju.edu.cn/wujx/paper/CNN.pdf

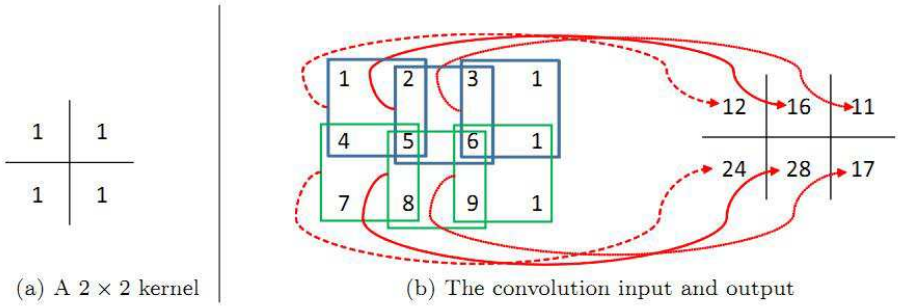

Figure 1.8 – Illustration of the convolution operation. [1]

of neurons from the previous layer.

Let us illustrate the convolution operation with a 3×4 matrix convolved by a size 2×2 convolution kernel as shown in Figure 1.8. As can be seen, the convolution operation repeats the computation of dot products between every possible window of the input matrix and the kernel from the top-left to the bottom-right, and then results into a 2×3 matrix.

As shown in Figure 1.9, suppose a convolution layer input tensor X has a size of $H \times W \times D$, and this layer contains D' filters with spatial size equals to $h \times w$. We will represent the filters of that layer as $\boldsymbol{f}$ by a 4D tensor in $R^{h \times w \times D \times D'}$. After convolving the input X by the filter $\boldsymbol{f}$ at every possible spatial location, we get an output tensor Y with size $(H - h + 1) \times (W - w + 1) \times D'$, in which the spatial size is smaller than that of the input when the filter size is larger than 1×1.

In some cases, we want the input and the output to have the same spatial size. As such, we usually *pad* each channel of the input layer with $\left\lfloor \frac{h-1}{2} \right\rfloor$ rows at the top and bottom, and $\left\lfloor \frac{w-1}{2} \right\rfloor$ columns on the left and right.

In Figure 1.8, the convolution is done at every possible spatial location, which corresponds to a *stride* $s = 1$. If $s > 1$, the convolution will be performed once every s pixels both horizontally and vertically which skips $s - 1$ pixels at every dot production. Note that the stride for the horizontal and vertical directions can be different.

In a more general case of convolution with padding and strides, we can compute

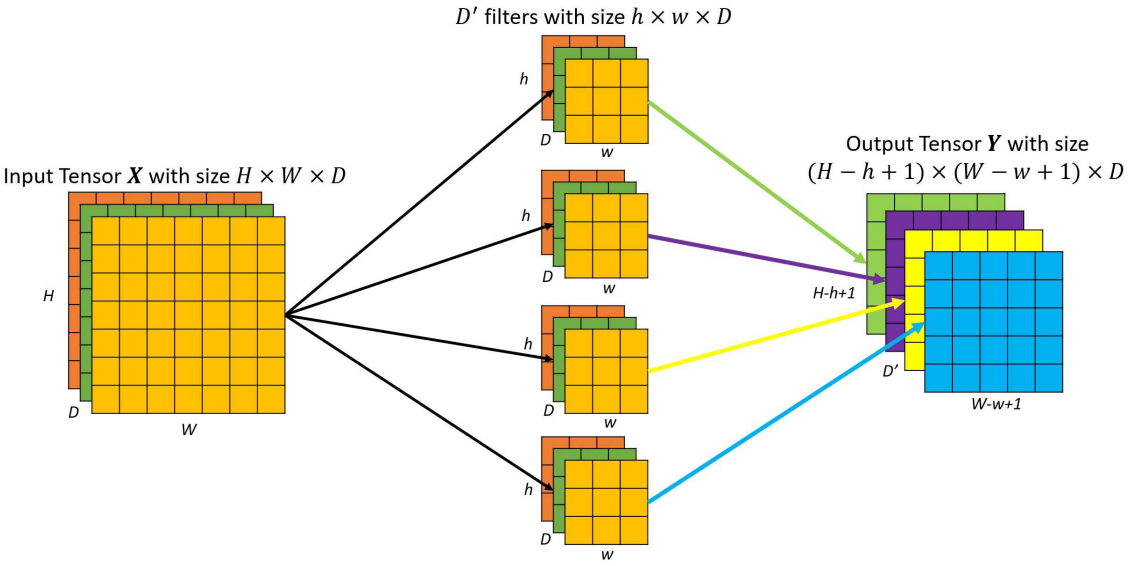

Figure 1.9 – A convolution layer convolves the input tensor X with size $H \times W \times D$ by using D' filters with spatial size $h \times w$, and results an output tensor with size $(H - h + 1) \times (W - w + 1) \times D'$. Here $H = W = 7$, $h = w = 3$, $D = 3$ and $D' = 4$.

the spatial size $H' \times W'$ of the output tensor Y by [140]:

$$H' = 1 + \left\lfloor \frac{H - h + P_h^- + P_h^+}{S_h} \right\rfloor \tag{1.59}$$

$$W' = 1 + \left\lfloor \frac{W - w + P_w^- + P_w^+}{S_w} \right\rfloor \tag{1.60}$$

where $(P_h^-, P_h^+, P_w^-, P_w^+)$ are the top-bottom-left-right padding of the input and (S_w, S_h) are the strides of horizontal and vertical directions.

The models reported in this thesis implement the simple case where the stride is 1 and no padding is used to see how a convolution operation is performed. As such, we can get an output tensor Y in $R^{H' \times W' \times D'}$, with $H' = H - h + 1$ and $W' = W - w + 1$. The convolution procedure for the k-th filter can be expressed as:

$$Y_{i,j,k} = \sum_{r=0}^{h-1} \sum_{c=0}^{w-1} \sum_{d=0}^{D-1} f_{r,c,d,k} \times X_{i+r,j+c,d} + b_k \tag{1.61}$$

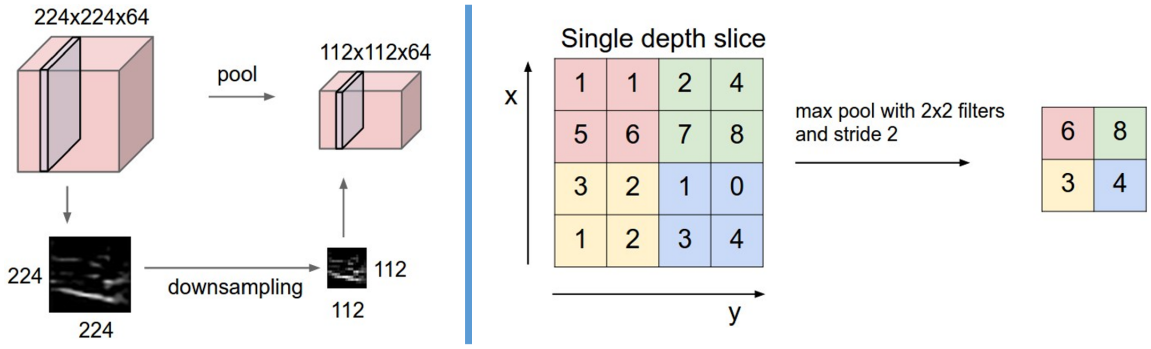

Figure 1.10 – A pooling layer which downsamples the input maps. On the right, the most commonly used "max pooling" layer with filter size 2×2 and a stride of 2. [3]

where $0 \leq i < H'$, $0 \leq j < W'$, $0 \leq k < D'$ and b_k is the bias of k-th kernel.

Recent studies [53, 127, 55] have shown that it is preferable to use small filters of size 1×1, 3×3 and 5×5 to build deep models than larger filters to construct shallow models.

Activation layer

After a convolution layer, an element-wise non-linear activation function is applied to each feature map to add non-linearity. The most commonly used functions are the sigmoid function, hyperbolic tangent function (Tanh), rectified linear unit (ReLU) and Leaky ReLU, etc. Please refer Section 1.3.3 for more details about these activation functions.

Pooling layer

After the activation operation, a pooling layer is often used to aggregate the feature maps. Its purpose is to progressively reduce the spatial size of the feature map to reduce the amount of parameters and computation in the network. It also adds robustness to local translation. Pooling is usually done by one of two types of pooling operators which are max pooling and average pooling over a 2D sliding window.

3. Image from: http://cs231n.github.io/convolutional-networks/

Let $X \in R^{H \times W \times D}$ be the input tensor to the pooling layer, and $h \times w$ the size of a pooling kernel. Assume that h divides H and w divides W, and strides equal to the kernel size, the output Y of pooling will be a tensor of size $R^{H' \times W' \times D'}$, where

$$H' = \frac{H}{h}, \quad W' = \frac{W}{w}, \quad D' = D. \tag{1.62}$$

A pooling layer operates upon X channel by channel independently. Within each channel, the feature map with size $H \times W$ is divided into $H' \times W'$ non-overlapping sub-regions, and the size of each sub-region is $h \times w$.The max pooling operator computes the maximum response of each sub-region by:

$$Y_{i,j,d} = \max_{0 \le r < h, 0 \le c < w} X_{i \times h + r, j \times w + c, d}, \tag{1.63}$$

and the average pooling operator computes the average response of each sub-region by:

$$Y_{i,j,d} = \frac{1}{hw} \sum_{0 \le c < h, 0 \le r < w} X_{i \times h + c, j \times w + c, d} \tag{1.64}$$

where $0 \le i < H'$, $0 \le j < W'$, and $0 \le d < D$.

In practice, the most widely used pooling setup is a kernel size of 2×2 and a stride of 2×2 as shown in Figure 1.10.

Fully-connected layer

Fully-connected layers are usually located at the end of the network. Each neuron k is connected to each element from the previous layer's output. If the output X of the layer that precedes the fully-connected layer is a tensor, a flatten operation is usually used to convert it into a feature vector $\boldsymbol{v}$. Then the resulting value h_k is computed by a dot product:

$$h_k = \boldsymbol{w}_k^T \boldsymbol{v} + b_k \tag{1.65}$$

where $\boldsymbol{w}_k$ are the weights connecting to previous layer's feature vector $\boldsymbol{v}$, and b_k is the bias term. Note that it is possible to implement a fully-connected layer as a convolutional layer by using filters who's spatial dimension matches that of the input feature maps.

Chapitre 2

Traffic Analytics with Low Frame Rate Videos

Résumé

In this chapter we investigate the possibility of monitoring highway traffic based on videos whose frame rate is too low to accurately estimate motion features. As demonstrated in our previous work [99], traffic activity is highly correlated to the 2D spatial texture features. CNN has the superior ability of extracting discriminate texture features directly learned from the data, so in this chapter, we proposed several different CNN models to segment traffic images into three different classes (road, car and background), classify traffic images into different categories (empty, fluid, heavy, jam) and predict traffic density without using any motion features. In order to generalize the model trained on a specific dataset to analyze new traffic scenes, we also proposed a novel transfer learning framework to do model adaptation.

This chapter is published as a journal paper in IEEE Transactions on Circuits and Systems for Video Technology, 2016.

Commentaires

The proposed methods were discussed by the Ph.D candidate, Pierre-Marc

Jodoin and Hugo Larochelle. The Ph.D. candidate built the Motorway dataset and did all the experiments in the paper.

The Ph.D candidate and Pierre-Marc Jodoin wrote the paper. Shao-Zi Li, Song-Zhi Su and Hugo Larochelle helped for revising the paper.

Traffic Analytics with Low Frame Rate Videos

Zhiming Luo
School of Information Science and Technology, Xiamen University,
Fujian Key Laboratory of Brain-like Intelligent Systems
Xiamen, Fujian, 361005 China
Département d'informatique, Université de Sherbrooke,
Sherbrooke, Québec, Canada J1K 2R1
zhiming.luo@usherbrooke.ca

Pierre-Marc Jodoin
Département d'informatique, Université de Sherbrooke,
Sherbrooke, Québec, Canada J1K 2R1
pierre-marc.jodoin@usherbrooke.ca

Shao-Zi Li
School of Information Science and Technology, Xiamen University,
Fujian Key Laboratory of Brain-like Intelligent Systems
Xiamen, Fujian, 361005 China
szlig@xmu.edu.cn

Song-Zhi Su
School of Information Science and Technology, Xiamen University,
Fujian Key Laboratory of Brain-like Intelligent Systems
Xiamen, Fujian, 361005 China
ssz@xmu.edu.cn

Hugo Larochelle
Département d'informatique, Université de Sherbrooke,
Sherbrooke, Québec, Canada J1K 2R1
hugo.larochelle@usherbrooke.ca

Keywords: Traffic density, convolutional neural network, video surveillance, transfer learning.

Abstract

In this paper, we investigate the possibility of monitoring highway traffic based on videos whose frame rate is too low to accurately estimate motion features. The goal of the proposed method is to recognize traffic conditions instead of measuring it as is usually the case. The main advantage of our

approach comes from its ability to process low frame-rate videos for which motion features cannot be estimated. Our method takes advantage of the highly redundant nature of traffic scenes which are pictured from a top-down perspective showing vehicles on a predominant asphalted road surrounded by background objects. Due to the limited variety of objects pictured in traffic scenes, our method gets to learn features that are specific to such images. With these features, our method is able to segment traffic images, classify traffic scenes, and estimate traffic density without requiring motion features. Different CNN models are proposed to segment traffic images in three different classes (road, car and background), classify traffic images into different categories (empty, fluid, heavy, jam) and predict traffic density. We also propose a procedure to perform transfer learning of any of these models to new traffic scenes.

2.1 Introduction

Video surveillance systems are often used for traffic monitoring and to characterize traffic load. Knowing in real time traffic conditions is a key element to help drivers avoid jams. To our knowledge, current camera-based traffic monitoring systems all rely on motion features. These features are typically estimated with optical flow, motion detection, and vehicle tracking [71, 159, 125, 84, 20, 73, 58, 59, 121]. Motion features are then used to count the number of vehicles on the road, estimate traffic speed, and recover global motion trajectories.

But these traffic monitoring systems all share a fundamental limitation: for it to work, they need high frame rate videos so motion features can be reliably extracted. Although a bit old, the survey by Kastriaki et al. [72] is explicit on that issue. Furthermore, fast and reliable object tracking is still a challenge for cluttered scenes, especially when vehicles are partly occluded [98].

Unfortunately, low frame rate videos are common as many large-scale camera networks cannot stream and store high-frame rate videos gathered by thousands of cameras. Instead, cameras are often configured to send one frame every second or so over the network. This is especially true for cameras transmitting over a cellular

network whose bandwidth may vary over time making the throughput unpredictable. One might also think of a non-stationary low-altitude UAV which can only take a limited number of images of a given road when it flies over it. In such cases, one can only analyze traffic based on a few unregistered images out of which no motion features can be extracted.

In this paper, we propose a method to recognize traffic activity instead of measuring it with motion features. Although traffic activity seams unavoidably bound to the dynamic of a video, we show in this paper that traffic activity is highly correlated to its 2D appearance. This is illustrated in Fig. 2.3. From these pictures, traffic dynamics (here empty, fluid, heavily packed and jam) can be inferred without the need of a video. This is because of the highly redundant nature of traffic scenes which are systematically made of cars and trucks pictured on top of a grayish road surrounded by background objects, typically vegetation, buildings, and sky.

The primary objective of our method is to estimate traffic density. In the process of doing so, we show that two other traffic analytics problems can be solved: traffic classification (classifying a scene as being empty, fluid, heavy, or jam) and image segmentation (segmenting the scene in road, vehicle and background). We show that these problems can be solved with different convolutional neural networks (CNN). Due to the highly redundant nature of traffic images, accurate results can be obtained with a surprisingly limited number of training images, especially when the training and testing images come from the same video camera. We also show how transfer learning can be used to further improve results in the case of a different camera perspective and/or a severe illumination variation.

2.2 Related Works

During the past few years, many computer vision methods have been proposed to analyze traffic videos, most of which relying on hand-crafted video features. In our previous work [99], we validated that a well-designed and well-trained 2D pattern recognition system can be used to assess traffic conditions, and we tested several traditional bag-of-visual-word based 2D texture features. But since 2012, the year when Krizhevsky et al. [80] demonstrated how convolutional neural networks could

significantly outperform traditional computer vision methods, the number of CNN papers devoted to video analytics grew exponentially.

In this section, we review both traditional traffic analytics methods as well as the CNN methods applied to traffic images and videos.

2.2.1 Methods for analyzing traffic

Computer vision methods for analyzing traffic can be roughly divided into two groups: the vehicle-oriented ones and the holistic approaches.

Vehicle-oriented methods assess traffic activity mainly by detecting and/or tracking vehicles. For these methods, vehicles are first localized with a background subtraction method [125, 71, 7, 129, 6, 89, 108] and tracked with a keypoint tracker [129, 6, 89], a Kalman filter tracker [71, 7], or a blob tracking method [125, 108]. Resulting tracks are bundled together in order to identify traffic lanes and/or the average traffic speed and/or the average traffic density [58, 125, 9]. Note that traffic density can also be obtained by counting the number of cars passing through a region of interest [59, 108].

Unfortunately, tracking simultaneously a large number of moving objects is still a challenge nowadays. As a solution, some methods focus on understanding the traffic flow from a macroscopic (or holistic) point of view and thus avoid tracking. The global representation of a scene is obtained by accounting for spatio-temporal features other than tracking and background subtraction. For example, Derpanis and Wildes [73] use 3D spatio-temporal filters based on the third derivative of a Gaussian function. Porikli and Li [116] use features from the MPEG compressed domain, i.e. the DCT coefficients and the motion vectors. Lee and Bovik [84] accumulates optical flow over time in order to get a temporal average vector field which is then used to classify traffic flow. Other methods [20, 32] analyze traffic flow as a dynamic texture classification problem. Chan and Vasconcelos [20] defined an autoregressive stochastic process over the spatial and temporal components while Derpanis and Wildes [32] associates different distributions to different traffic flow statuses in a spatio-temporal orientation domain.

2.2.2 Video-based CNN methods

Convolutional neural networks (CNN) [82] are biological-inspired hierarchical multi-layer neural networks. Recently, methods using a CNN have established state-of-the-art performances in numerous computer vision tasks such as image classification [80, 127], object detection [54, 44], and action recognition [66, 126].

CNNs have also been applied in the field of traffic analytics. Alvarez et al. [5] uses a CNN for segmenting road images taken by an on-board camera. At first, a CNN model is trained on a general-domain dataset (LabelMe) to classify pixels into sky, vertical area and ground pixels. Then, online learning of textures allows to further refine the segmentation. The final road segmentation is obtained by combining offline and online features following a naive Bayesian framework. Brust et al. [17] improved the approach by adding a spatial prior to the CNN model. Dong [34] proposed a semi-supervised method for training a CNN model used to classify front-view vehicles into different categories. The car dataset proposed by Yang et al [153] as well as ImageNet were used to train the model. Jang [64] also used CNN to classify image patches into different vehicle categories. Zheng [158] adopted CNN to classify single vehicle or two vehicles in virtual coil region which aims for vehicle counting. Some other traffic-related CNN models were proposed for traffic sign recognition [69] and vehicle logo recognition [61]. Beside these CNN based deep learning methods, Lv et al. [104] proposed an auto-encoder based deep learning method to learn spatial and temporal correlated features for traffic flow prediction.

Indeed, traffic density flow status also can be accessed by counting the number of vehicles. In the last couple of months, many CNN based vehicle detectors have been developed [19, 160, 156] that provide excellent detection results at near real time. But the main inconveniences with these approaches are threefold. First, for it to work, the training and testing data must come from cameras with almost the same configuration (*same height, angle, and illumination*). As such, these approaches do not generalize well to a wide variety of cameras. Second, these methods can only detect vehicles at a certain scale and are doomed to fail on small vehicles seen from far away which are very common in real traffic scenes as showed in Fig. 2.3, 2.4 and 2.6. Third, none of these methods performed well in traffic jams as strong occlusion makes these methods fail. Although deep learning vehicle detectors could potentially work

well under difficult situations, they nonetheless require a very large training dataset (like ImageNet but for traffic scenes) that nobody has released yet.

Compared with our previous work [99] of using traditional 2D texture features and pre-trained CNN features for traffic flow status classification, in this paper we proposed several different CNN models for traffic density estimation.

2.3 Our Method

2.3.1 Estimating traffic density

As mentioned by Darwish and Bakar in their recent extensive survey [31], there are many different ways of measuring traffic density, many of which depending on the device used to record traffic. Some authors refer to traffic density as the number of vehicles recorded over a certain period of time (without taking into account the length of the road) [2, 106], others use the number of vehicles per unit of road [110, 74] while others use a definition as the fractions of the road covered by cars [6, 41]. As such, there is no unique gold standard measure of traffic density.

Since our method has only access to traffic images and low frame-rate videos, the speed at which vehicles are traveling cannot be measured. As a consequence, we define traffic density as being the percentage of the road being occupied by vehicles, i.e.

$$Density = \frac{Vehicle}{Vehicle + Road} \tag{2.1}$$

where "$Road$" accounts for the visible surface of the road and "$Vehicle$" for the surface occupied by vehicles which is identical to the one used [6, 41]. Compared with the most widely used one provided by Kerner [74] for which traffic density is the number of vehicles per unit length of a road [vehicles/km], there is an obvious correlation between the number of vehicles on the road and our vehicle-to-road ratio since as the number of vehicles increase on the road, the vehicle-to-road ratio unavoidably increases. This statement is rigorously true when the camera is looking at the road from the side (as in [110]) but has to be considered with care when the camera is looking at the road from a perspective angle. In the latter case, depending on

the camera elevation angle, the vehicle will suffer from a partial volume effect and Eq.(2.1) will give a traffic density value slightly higher than its actual value.

There are different ways of estimating Eq.(2.1), the simplest being to count the number of vehicle and road pixels in a given image. Doing so converts the problem of traffic density estimation into a problem of scene labeling. Alternatively, one can directly estimate traffic density by "recognizing" it, i.e. by assigning it a low value when a large grayish area is seen (the road) and a large value when a relatively high frequency texture is measured (cars and trucks on the road). Doing so converts the problem of traffic density estimation into a regression problem of mapping texture features to a traffic density value ranging between 0 and 1.

In this paper, we propose three different CNN models for estimating traffic density. The first CNN performs a global regression. It converts the entire image into a feature vector and then use a support vector regression (SVR) to predict traffic density. The second and third CNN models are local patch-based models. The second model aims at segmenting the scene into three semantic categories, i.e. road, vehicles and background. The third model uses regression to estimate the proportion of road, vehicle and background in local patches.

In the following subsections, we introduce a generic CNN architecture upon which our models are built. We then provide details of our CNN models and show how they estimate the traffic density of Eq.(2.1).

2.3.2 Generic CNN architecture

Convolutional neural networks usually consist of a series of convolutional layers, activation layers, pooling layers, and fully-connected layers [82]. The main difference between CNN models often comes from the number of layers and the way they are connected together.

Convolutional layers These are made of N filters which convert X input image channels (or features maps) into N new feature maps. Convolution layers are typically organized in a cascaded fashion such that the output feature maps of one layer is the input of the subsequent one. The convolution of feature maps (or the input image for

the first layer) can be summarized by the following linear operation:

$$(f_k)_{ij} = (w_k * x)_{ij} + b_k \tag{2.2}$$

where w_k and b_k are the filter and bias of the kth kernel, $*$ is a convolution operator, and$(f_k)_{ij}$ is the kth output feature at spatial location (i, j). At each position (i, j), the output from convolution operations are stacked together leading to N feature maps. Recent studies [53] show that a deep architecture will small filters are to be preferred to shallow architectures with large filters.

Activation layers After a convolution layer, a non-linear activation function is applied to each feature map neuron. The most commonly used functions are the sigmoid function, hyperbolic tangent function, and rectified linear unit (ReLU).

Pooling layers After the activation operation, each feature map f_k gets to see its pixels aggregated together. This is done by computing the average or maximum value over a 2D sliding window. This layer is known for adding robustness to local translation, and reduces the feature map size when the stride of the sliding window is larger than 1.

Fully-connected layers Such layers are usually located at the end of the network. Each neuron k is connected to each output x_i of the previous layer. The resulting value y_k is computed by a dot product:

$$y_k = \sum_i w_{ki} x_i + b_k \tag{2.3}$$

where w_{ki} is the weight connecting to previous layer's neuron i, and b_k is the bias term. Note that it is possible to formulate a fully-connected layer as a convolutional layer, by using filters who's spatial dimensional matches that of the input feature maps.

2.3.3 Global regression model

Previously, Krizhevsky et al. [80] successfully applied a CNN on a large scale image classification task (ImageNet), and their approach quickly became the de facto solution for many computer vision tasks. In a 2014, Razavian et al. [124] showed that for a wide variety of problems, an ImageNet pre-trained CNN model can be successfully used as a generic feature extractor. They showed that previous state-of-the-art codebook and part-based methods can be outperformed by using the 4096 dimensions of the penultimate fully connected layer as a feature vector together with a linear SVM classifier.

Following Razavian et al.'s work, we also use a pre-trained ImageNet CNN model to characterize the dynamic content of a traffic image. The feature vector extracted from the pre-trained CNN model is fed to a support vector regression *(SVR)* method to estimate traffic density. This model is regarded as our global regression model.

In our implementation, we use the ImageNet pre-trained VGG-VeryDeep-16 model *(VGG-16)* [127] to compute image features for our traffic images. This model consists of five consecutive convolutional blocks (Conv-1 to Conv-5) and three fully connected blocks (FC-6 to FC-8). Each convolution block consist of several convolution layers, activation layers and pooling layers. In this paper, we evaluate the performance of the features extracted from block Conv-4 to FC-7. The corresponding layers in the original VGG-16 are *relu4_3*, *relu5_3*, *relu6* and *relu7*.

The VGG-16 model was originally designed to classify images of size 224×224 into 1000 classes. However, it can also be used as a feature extractor for images of arbitrary size. After converting FC-layers in the VGG-16 model into corresponding convolution layers, the output of each layer i is a 3D feature map of size $m_i \times n_i \times d_i$ where $m_i \times n_i$ is the spatial resolution which depends on the model architecture and the input image size. In order to make sure the feature vector of each layer i has the same dimension d_i regardless of the input image size, we use average pooling over the $m_i \times n_i$ spatial regions to get the d_i features of layer i. As in the VGG-16 model, the features dimensions of Conv-4, Conv-5, FC-6 and FC-7 are 512, 512, 4096 and 4096.

Once the image features are computed, a linear SVR as well as 3 different kernel SVR *(Hellinger, Chi2 and HIK)* were trained and tested for traffic density estima-

tion. These kernels are $k_{hellinger}(\vec{v},\vec{v}') = \sum_{i=1}^{N}\sqrt{v_i v_i'}$, $k_{chi2}(\vec{v},\vec{v}') = \sum_{i=1}^{N}\frac{2v_i v_i'}{v_i+v_i'}$ and $k_{hik}(\vec{v},\vec{v}') = \sum_{i=1}^{N} min(v_i, v_i')$, where $\vec{v}$ and $\vec{v}'$ are image features.

2.3.4 Local CNN models

In this work, we also investigate two CNN models that, instead of producing a single prediction for the full input image, make predictions locally by performing computations on several small patches extracted across the input image. We refer to these models as SegCNN and RegCNN.

SegCNN

As mentioned earlier, traffic density can be estimated by counting vehicle, road and background pixels in a traffic image. In order to do so, we propose a local CNN model whose goal is to segment each pixel into 3 semantic categories (Road, Vehicle, Background), which we call "SegCNN".

SegCNN predicts the category (Road, Vehicle, or Background) of each pixel by processing the 31×31 image patch centered on it. The SegCNN model contains 4 convolutional layers (Conv-1 to Conv-4) and 2 fully connected layers (FC-5 to FC-6). For Conv-1, we use 32 filters of size $7 \times 7 \times 3$ while for Conv-2, Conv-3 and Conv-4 we use 32 filters of size $7 \times 7 \times 32$. The Rectified linear unit (ReLU) is used as activation function for Conv-1 to Conv-4. Also the first two convolutional layers are followed by a 2×2 max pooling layer. All convolution and pooling layers use a stride of 1.

The fully connected layer FC-5 has an output feature map of size 64 and the output of FC-6 is 3, for the 3 scores that we predict (Background, Vehicle and Road) for each pixel. A softmax function is applied to the 3 scores to compute 3 probabilities. At test time, a pixel label prediction is made based on the category with maximum probability. The negative log-likelihood based on the softmax function is used for training the CNN's parameters. Details for the SegCNN model are provided in Table. 2.1.

Due to perspective, vehicles within the same image can have very different sizes. In order to compensate for this effect, we made our approach multi-scale as illustrated in Figure 2.1 at the testing procedure. As one can see, the input color image is resized

Table 2.1 – Architecture details of the SegCNN model

Layer	1	2	3	4	5	6
Stage	conv	conv	conv	conv	FC	FC
Input size	31×31	25×25	19×19	13×13	7×7	1×1
Filter size	7×7	7×7	7×7	7×7	-	-
Conv stride	1×1	1×1	1×1	1×1	-	-
Pooling method	max	max	-	-	-	-
Pooling size	2×2	2×2	-	-	-	-
Pooling stride	1×1	1×1	-	-	-	-
Padding size	-	-	-	-	-	-
#Channels	32	32	32	32	64	3

into 3 scales (0.5, 0.75, 1), each of which is being processed by our SegCNN model. The resulting probability maps (here illustrated with a red/green/blue image) are then upsampled to the input reference size. A max pooling operation across all output channels is used to get the final segmentation result. For each pixel, the predicted label is set to the category which has the maximum value.

To train the SegCNN model, a two step procedure is used. During the first step, we randomly sample 1500 patches from each training image. Stochastic gradient descent is adopted to update the model's parameters with a momentum of 0.9. The model is trained for 60 epochs with a batch size of 256. The learning rate is initialized at 0.001 and then decreased by a factor of 10 after every 20 epochs.

Once the first step is done, each training image is used as input for the model to compute its corresponding output. The model's parameters are updated based on a softmax loss computed between the whole image groundtruth and the CNN output. At this step, we refine the model with 40 epochs with a batch size of 1 (one image). The learning rate is initialized at $1e{-}5$ and decreased to $1e{-}6$ after 20 epochs.

RegCNN

We observed that labeling each pixel into a specific semantic category with the SegCNN model is a time consuming procedure. Since we need a prediction for each pixel, the total number of patches to process equals the number of pixels in the input image. As a workaround, we propose a patch-based regression model which

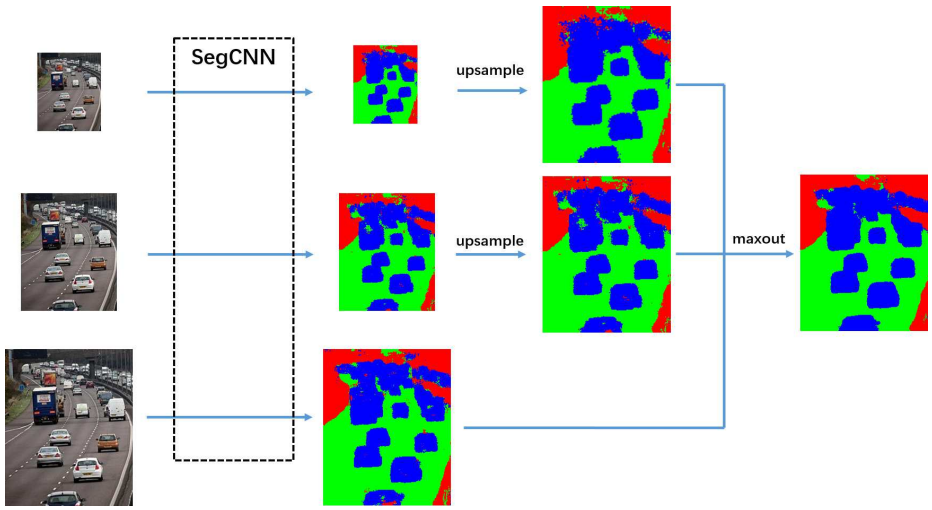

Figure 2.1 – A illustration of the multi-scale information combining procedure. The input image is resized into 3 scales, each of which being fed to our SegCNN model. The output are then upscaled and combined following a maxout procedure.

processes a much smaller number of partly overlapping patches. We refer to this model as RegCNN. This model is based on the observation that that Eq.(2.1) could be computed using predictions for the proportions of Road and Vehicle over local patches.

Thus, instead of predicting pixel-wise semantic labels like SegCNN, the goal of RegCNN is to predict, for any given patch, the proportion of vehicle, road and background pixels inside of it. Then, a post-processing procedure aggregates the road and vehicle proportions across every patch and computes traffic density.

Let's consider a given $m \times n$ input image which we divide into a series of $s \times s$ patches. If the tiling of these patches follows a stride of s pixels, we get a total of $(m/s) \times (n/s)$ non-overlapping contiguous patches. Without loss of generality, one can count the total number of vehicle and road pixels in that image (aka $Vehicle$ and $Road$ in Eq. (2.1)) by summing across all non-overlapping patches the total number of vehicle and road pixels located inside of it. Traffic density can thus be computed as :

$$Density = \frac{Vehicle}{Vehicle + Road} = \frac{\sum V_p}{\sum V_p + \sum R_p} \tag{2.4}$$

where V_p and R_p are the number of vehicle and road pixels inside patch p.

If instead we use a stride d that is smaller than s and pad the image border with $s-d$ background pixels, we get a total of $((m+s-d)/d) \times ((n+s-d)/d)$ overlapping patches. The sum of the total number of vehicle (V_p) and road pixels (R_p) contained in all overlapping patches is s/d times larger (in expectation) than in the original image. However, since the s/d is the same multiplication factor for $\sum V_p$ and $\sum R_p$, the traffic density computed in Eq.(2.4) remains the same.

Since predicting the total number of road and vehicle pixels in a patch is ill suited for regression, we instead predict the road and vehicle ratios V_p^{ratio} and R_p^{ratio} by dividing V_p and R_p by the patch size $s \times s$. Again, since the normalization factor is constant for V_p and R_p, we may compute traffic density as follows:

$$Density = \frac{\sum V_p^{\text{ratio}}}{\sum V_p^{\text{ratio}} + \sum R_p^{\text{ratio}}}. \tag{2.5}$$

In additional, our RegCNN also predicts the background category's percentage ratio B^{ratio} for each patch.

RegCNN processes patches of size 32×32. The CNN's architecture contains 3 convolution layers (Conv-1 to Conv-3) and 2 fully connected layers (FC-4 to FC-5). Conv-1 to Conv-3 use 32, 32 and 64 filters of kernel size $5 \times 5 \times 3$, $5 \times 5 \times 32$ and $5 \times 5 \times 32$ with zero padding, and Rectified linear unit (ReLU) is used as activation function. After the activation function, each layer is followed by a 3×3 average pooling layer of stride 2×2. The output size of the fully connected layer FC-4 is 128, while it is 3 for FC-5, one for each predicted ratio : V^{ratio}, R^{ratio} and B^{ratio}. Details of the whole RegCNN architecture are given in Table 2.2. Note that a least square error function is used as the loss function for training the model.

When using RegCNN to predict the whole image, we first pad each border with $s-d$ ($s = 32, d = 8$) zero values, and the fully connected layers are reshaped as convolution layers with stride 1×1. Since RegCNN contains 3 pooling layers all with a 2×2 stride, the total stride of the network is 8×8. Consequently, the predicted ratio maps of RegCNN are 8 times smaller than the input image. Also, we empirically noticed that as opposed to SegCNN, RegCNN does not benefit from a multi-scale configuration as in Fig. 2.1. This is why RegCNN comes with a single-scale configuration.

Table 2.2 – Architecture details of the RegCNN model

Layer	1	2	3	4	5
Stage	conv	conv	conv	fully	fully
Input size	32×32	16×16	8×8	4×4	1×1
Filter size	5×5	5×5	5×5	-	-
Conv stride	1×1	1×1	1×1	-	-
Pooling method	average	average	average	-	-
Pooling size	3×3	3×3	3×3	-	-
Pooling stride	2×2	2×2	2×2	-	-
Padding size	2×2	2×2	2×2	-	-
#Channels	32	32	64	128	3

Training the RegCNN model

We use a two-step training procedure very similar to the one described for SegCNN. The only difference comes with the learning rate which we fix to 0.001 for the first step and 0.0001 for the second step. We also use 200 epochs for the first step and 60 epochs for the second step.

2.4 Transfer Learning

The ImageNet challenges have highlighted that CNNs are very successful models for computer vision when trained on very large quantities of data. Unfortunately, at the time of the writing of this paper, the number of annotated traffic images that could be used to train our CNNs was very limited. And with the burden of labeling thousands (if not millions) of images like those in Fig. 2.3, it is likely that no such dataset will ever be released. Yet, the ability to adapt to conditions different from those in the training set is important for an approach such as our's.

Fortunately, as will be shown in the results section, one only needs a limited number of images to successfully train our models when the training and testing images come from the same camera or when they show similar weather conditions and perspective on the traffic (c.f. Fig. 2.3).

That said, it is a challenge to process test images whose perspectives on traffic, weather conditions, or global illumination are very different than in the training set.

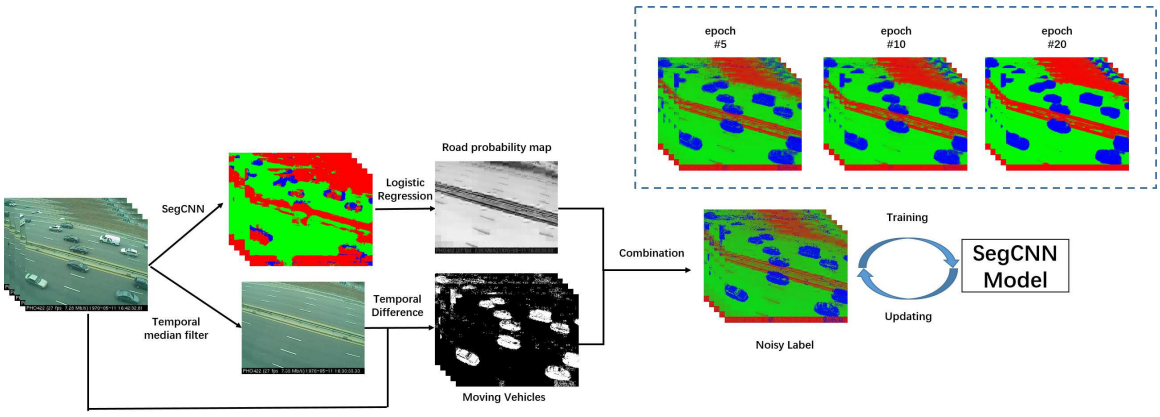

Figure 2.2 – Our transfer learning procedure includes two main steps. The first step is a noisy label initialization which combines the output of the original SegCNN model, color information and a rough motion detection. The second step adjusts SegCNN's parameters and update noisy labels with a feedback loop.

One way of working around this problem is to provide our model with additional training information on the content of the test video. For SegCNN, this is done following a two-step transfer learning procedure. The first step combines the output of the SegCNN network with color information and rough motion detection. Since this information is usually imprecise, we refer to it as the noisy labels. In a second step, we leverage these noisy labels and update the model's parameters following a feedback loop. The whole procedure is shown in Fig. 2.2.

As for the global regression model and the RegCNN model, we use the results of SegCNN (after transfer learning) as a groundtruth to update their parameters.

Step 1: Noisy labels initialization

The label initialization procedure is shown on the left-part of Fig.2.2. Given images from a new camera, we first select N frames $(F_1, ..., F_N)$ spanning over a certain period of time (typically a few minutes). Because road areas share a similar structure and color information regardless of the viewpoint of the camera and the weather condition (typically a flat grayish surface) the SegCNN road prediction is usually accurate (c.f. the green area in the top-left of Fig.2.2). This allows us to produce a rough initialized

probability map of where the road is in each frame F_i.

As there are still some missing road areas, a logistic regression model is trained on the initialized road probability maps to recover those missing areas. We randomly sample 5000 pixels in each frame to train the logistic regression model by using RGB and Lab color values. Assuming the camera is fixed, we first use the logistic regression model to compute a road probability map for each frame and then average these probability maps to get a road probability map (RPM):

$$RPM = \frac{\sum_i^N Logistic_{\text{road}}(F_i)}{N}. \tag{2.6}$$

Since surveillance videos are usually captured by static cameras, we may also get a rough estimate of where moving vehicles are with a simple median background subtraction method [28]. Note that background subtraction results are likely to be noisy since the framerate is very low. In order to reduce false detections due to background moving objects (typically trees in the wind), we only keep foreground pixels located inside the road area. We then assign those foreground pixels a 0.9 probability of being a vehicle pixel, and 0.1 to the other pixels:

$$P_{\text{vehicle}} = \begin{cases} 0.9 * (RPM > 0.5) & \text{foreground} \\ 0.1 & \text{others.} \end{cases} \tag{2.7}$$

The corollary of this equation is that the probability for a pixel located in the RPM to be part of the road and not a vehicle could be estimated as:

$$P_{\text{road}} = (1 - P_{\text{vehicle}}) * RPM. \tag{2.8}$$

Once P_{Vehicle} and P_{road} are computed, since the pixel's likelihood probabilities must sum up to 1, the background likelihood probability is :

$$P_{\text{back}} = 1 - P_{\text{vehicle}} - P_{\text{road}}. \tag{2.9}$$

Step 2: Updating the model

Given $P_{\text{back}}, P_{\text{vehicle}}$, and P_{road} (our noisy probability labels) the next step is to update the parameters of SegCNN over every frame F_1 to F_N. For doing so, we use a cross-entropy loss to update the CNN parameters:

$$L = -\frac{1}{N}\sum_{i=1}^{N}\sum_{c=1}^{3} p_{i,c}\log(q_{i,c}) \tag{2.10}$$

where N is the number of training pixels, $p_{i,c}$ is the noisy probability map computed with Eq.(2.7) to (2.9) and which we use as groundtruth, and $q_{i,c}$ is the predicted probabilities for a given category.

Following a feedback loop, after each epoch t, we update the noisy likelihood maps $p_{i,c}^{t-1}$ given the newly predicted probability label $q_{i,c}^{t}$:

$$p_{i,c}^{t} = (1-\alpha)p_{i,c}^{t-1} + \alpha\mathbb{1}(q_{i,c}^{t}) \qquad \forall c \in \{\text{road, vehicle, back}\} \tag{2.11}$$

where $\mathbb{1}(q_{i,c}^{t})$ is an indicator function giving 1 to the category label with largest probability and 0 to the others. In the upper-right part of Fig. 2.2, we give the updated probability maps q^t for epoch 5, epoch 10 and epoch 20.

Details of the training procedures

For each video, the first 400 frames are used for transfer learning and the remaining frames are used for testing. After initializing the noisy labels, we train the SegCNN model for 30 epochs by using the proposed training and updating feedback loop with a learning ration of 0.0001. Since the 3 categories data are unevenly distributed, we randomly sample at each epoch 60,000 training pixels of each category to compensate for this issue.

Once transfer learning is done for SegCNN, we use the SegCNN results as groundtruth for training the global regression model and RegCNN model. For the global regression model, we take the feature of the Conv-4 layer and with the kernel because this combination has the best performance (c.f. Section 2.5). And for the RegCNN model, we train the model for 30 epochs with a learning rate of 0.0001.

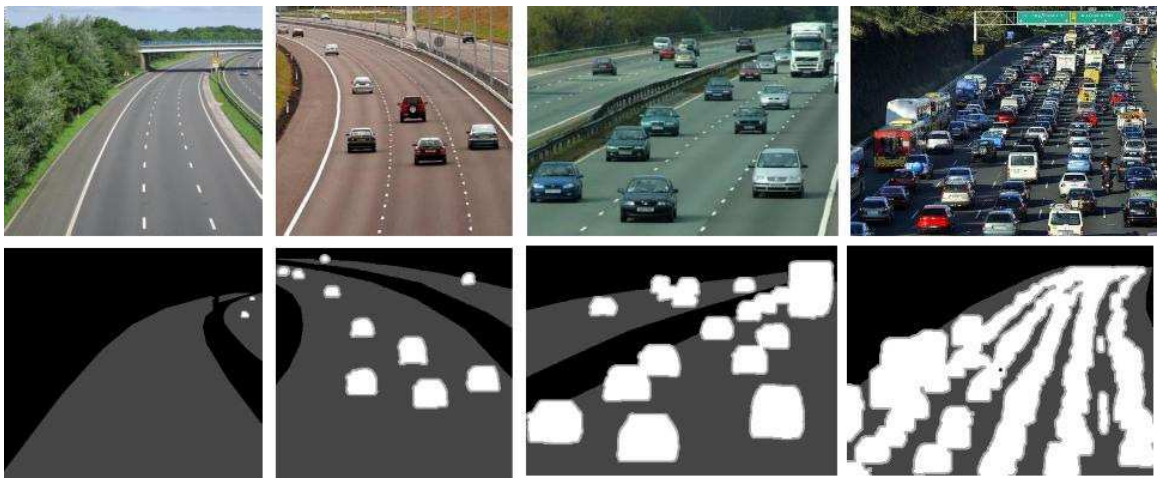

Figure 2.3 – Images from our Motorway Dataset showing different traffic density. Each image has been manually segmented into 3 semantic categories ($Road, Car, Background$).

2.5 Traffic density estimation results

In this section, we evaluate our models without transfer learning on two traffic datasets: the Motorway dataset [99] and the UCSD traffic dataset [20]. The Motorway dataset contains images from different cameras while the UCSD dataset contains several videos from one single camera. Experimental results of our transfer learning framework applied on other traffic videos are presented in the next section. The MatConvNet [138] toolbox was used for all our CNN model implementations.

2.5.1 Motorway Dataset

This dataset contains 400 images taken from 400 highway cameras deployed in the UK. These images have been divided into 4 traffic categories, namely empty, fluid, heavy and jam. Each pixel in the dataset has been manually labeled into 3 different semantic categories, i.e.: Vehicle, Road and Background. Images from that dataset are shown in Fig. 2.3.

All of our models have been trained on 200 randomly-selected images and tested on the remaining 200. We used the Mean Absolute Error (MAE) to compare the

estimated traffic density to real traffic density, computed with Eq.(2.1). The metric is as follows:

$$MAE = \frac{100}{n} \sum_{i=1}^{n} |d_i - \hat{d}_i| \tag{2.12}$$

where d_i is the traffic density for image i, n is the total number of test images, and 100 is to express the result as a percentage.

Global regression model

the features computed from the VGG-16 model are $L2$ normalized into unit length. The LibLinear toolbox [39] is used for training the SVR models with a fixed slack parameter of $C = 1$. Because Chi2 and HIK kernels are additive kernels, we used the Vedaldi et al. [139] method to improve efficiency. With their method, features are mapped into a related high dimensional space in which kernel-SVR can be approximated by a linear SVR which is more effective.

The MAE (and standard deviation) of 10 repeated experiments are shown in Table.2.3. As can been seen, the error does not vary much from one kernel to another and from a layer to another. That being said, the best results were obtained with features extracted from Conv-5 together with the Hellinger kernel. The MAE for this combination is 5.83% which is very low considering the fact that the traffic density difference between the images of Fig. 2.3 is roughly 25%. As a rule of thumb, a 5.83% MAE correspond to a missing car in the third image of Fig. 2.3. Since we do not account for video features (each video comes with only one frame) those results provide a good indication that traffic is strongly correlated with its 2D appearance as we hypothesized at the beginning of this work.

Table 2.3 – The MAE of the global regression model with different kernels on the Motorway dataset

	Conv-4	Conv-5	FC-6	FC-7
Linear	7.25±0.39	7.70±0.51	7.95±0.43	7.79±0.46
Hellinger	6.75±0.42	5.83±0.32	6.33±0.42	6.30±0.47
Chi2	6.10±0.56	6.10±0.53	6.68±0.48	6.38±0.42
HIK	6.00±0.55	6.81±0.42	7.01±0.27	6.63±0.52

SegCNN

Since SegCNN is a segmentation model, in addition to the traffic density MAE, we also compute the accuracy of the segmentation results (that is the number of correctly labeled pixels divided by the total number of pixels). The quantitative results of 10 repeated experiments are showed in Table 2.4. As can be seen, the second training step slightly improves the overall performance while the use of multiples scales improves performances even more. Note that the MAE obtained by this model is almost half as small as the global regression model. Fig.2.4 shows some of the segmentation results computed by our CNN model.

Table 2.4 – The Accuracy on Motorway dataset of SegCNN model

	Accuracy	MAE
First training step	86.53±0.49	4.27±0.40
Second training step	87.53±0.47	3.85±0.33
Multi-Scale	89.10±0.30	3.63±0.34

RegCNN

The MAE of this model is presented in Table 2.5. Here again, the second training step is helpful to improve performance as it reduces the MEA by almost a factor of 2. We also plot the estimated ratio maps of each category in Fig.2.5. Note that the blurry aspect of these maps comes from the fact that we had to resize it to the original image size (c.f. Section 2.3.4 for why the maps are 8 times smaller than the original images). From that figure, we can see that our RegCNN model's results are consistent with the *Vehicle*, *Road* and *Background* categories.

Note that RegCNN only needs 4 ms to compute the traffic density ratio of a 256×256 image on a Nvidia GTX 970 GPU, while our SegCNN models needs 24 ms for single scale and 59 ms for multi-scale. This makes our RegCNN model between 5 to 14 times faster than SegCNN (at the cost of a slight decrease of MAE : 4.5% for RegCNN and 3.63% for SegCNN).

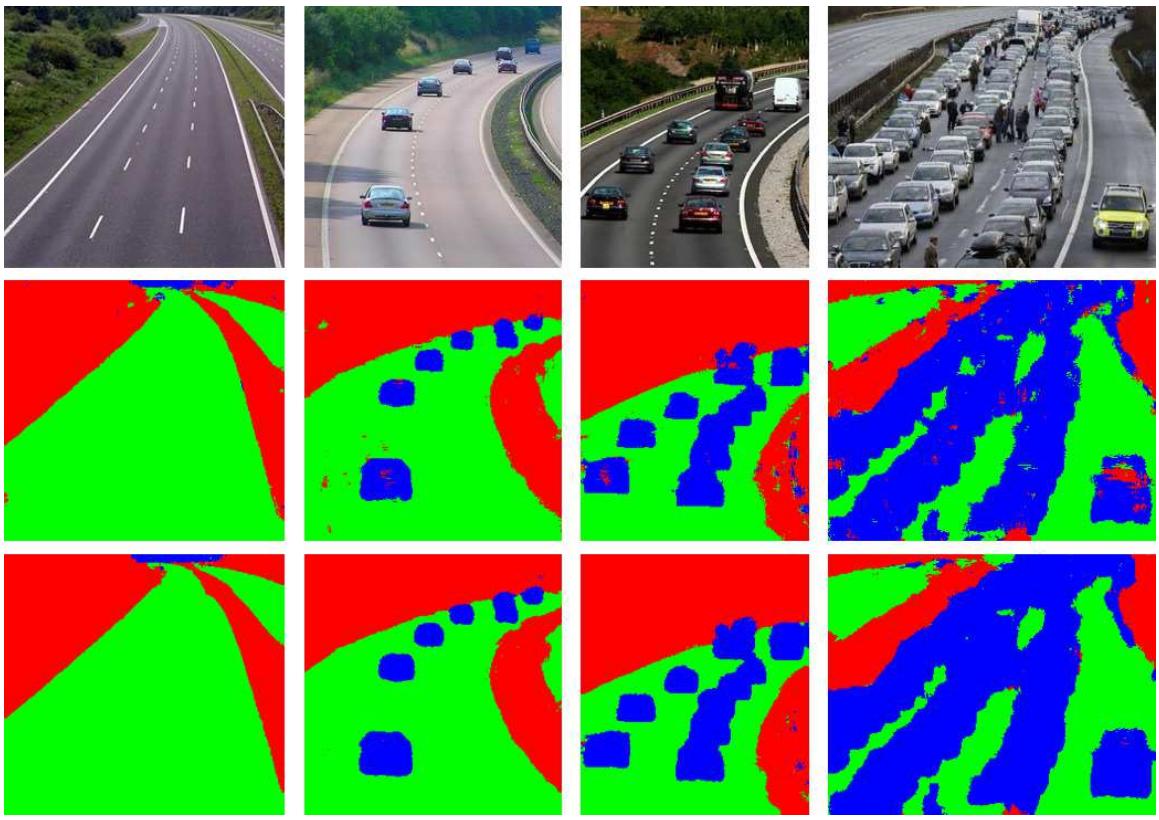

Figure 2.4 – Results of our SegCNN model. Second row: results for a single-scale model. Third row: results after combining multiple scales.

Comparison with other methods

In Table 2.6, we compared our approaches with three different methods. The first one uses bag-of-visual-word texture features plus a SVR model to estimate the traffic density [99]. The second one is similar to [99] as it implements a Locality-constrained Linear Coding (LLC) [142] with 1024 visual words and a HIK SVR kernel. The third method is the fully convolutional network model (FCN) [97] which is a very recent state-of the-art CNN based image segmentation method. As showed in Table 2.6, the FCN model which has a much deeper structure and more parameters gives a slightly higher pixel segmentation accuracy and lower MAE than our SegCNN model. As for the BOVW+SVR and LLC+HIK, their results are far worst than the other

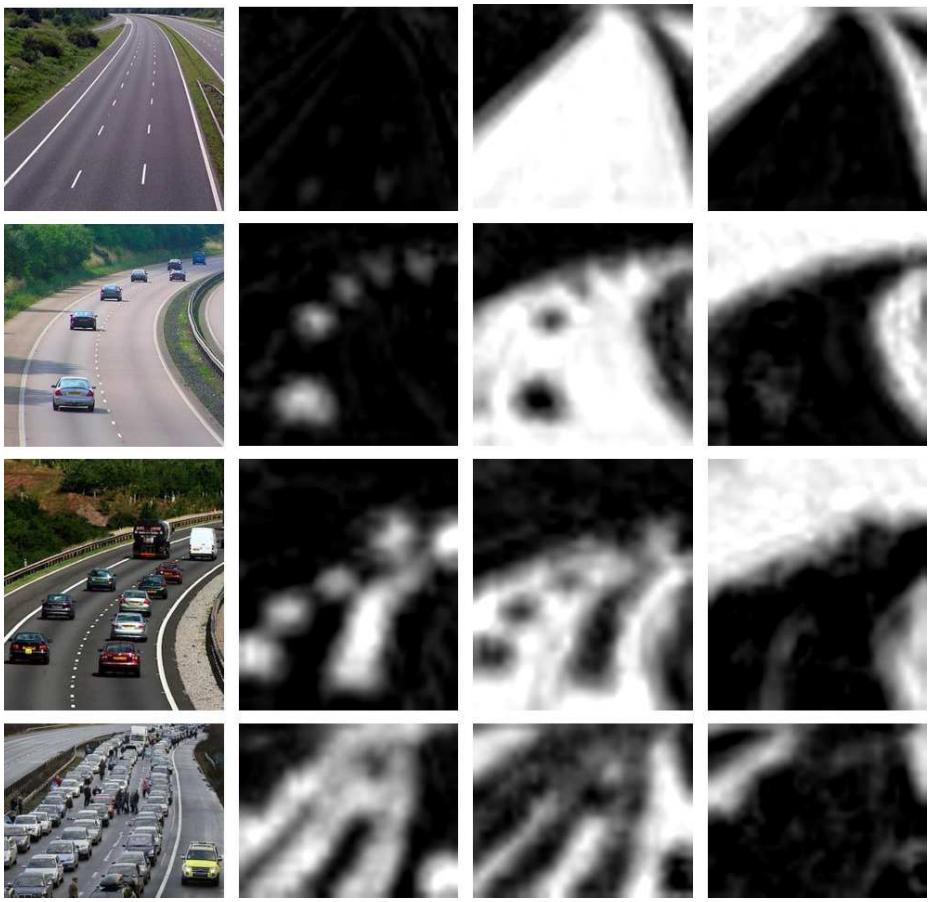

Figure 2.5 – Visualization of the RegCNN model's estimated ratio maps. The second column is the ratio map of the vehicle category, the third column is the road category and the the last column is the background category.

approaches.

2.5.2 UCSD Traffic Dataset

The goal of the previous experiments was to measure how good our models are at estimating traffic density from images pictured by different cameras. The goal of the UCSD dataset is to measure how accurate our methods are when trained and tested on videos obtained from the same camera.

The USCD dataset contains 254 highway video sequences filmed at different periods of the day and under different weather conditions by the same camera. Each

Table 2.5 – MAE obtained by RegCNN on the Motorway dataset

	MAE
Fisrt training step	8.71±1.54
Second training step	4.50±0.37

Table 2.6 – Comparison results on Motorway dataset

Method	Accuracy	MAE
BOVW+SVR[99]	-	6.67±0.33
LLC+HIK[142]	-	5.83±0.32
RegCNN	-	4.50±0.37
SegCNN-MS	89.10±0.30	3.63±0.34
FCN-8s[97]	91.07±0.54	3.11±0.53

video comes with a frame rate of 10 fps and a total of 50 $320 * 240$ frames. These 254 videos have been divided into three classes based on the number of vehicles and overall traffic speeds: 165 light traffic videos, 45 medium traffic videos, and 44 heavy traffic videos. Since we only focus on low frame rate videos, we only process 1 frame out of 6 (~ 1.7 *fps*), which accounts for 8 frames $(f_1^k, f_2^k, ..., f_8^k)$ per video v^k. We followed the training/testing protocol provided by the authors of this dataset [20]. The results are computed based on a 4-fold cross validation for which 75% of the dataset are used for training and the rest 25% for testing. Fig.2.6 shows 3 representative frames from that dataset.

Since the training procedure of SegCNN and RegCNN requires pixel-accurate labels and that the UCSD dataset contains none, we only evaluated our global regression model on this dataset.

In order to train the global regression model, we set the traffic density of each frame as follows: Light=0.1, Medium=0.5, and Heavy=0.9. At test time, given a test video, the global regression model predicts the density of its 8 frames and then average them out. As for the Motorway dataset, the features are L2 normalized into unit length. Two thresholds (T_{hm}, T_{ml}) computed on the training dataset are used to classify the test videos into the three classes. T_{hm} is the threshold separating the medium and heavy traffic classes, while T_{ml} is the threshold separating the medium

Figure 2.6 – Frames from the UCSD traffic dataset showing different traffic density: Light(left), Medium(middle) and Heavy(right).

and light traffic classes. The equation for computing (T_{hm}, T_{ml}) are as follows:

$$T_{hm} = \frac{\overline{h} + \overline{m}}{2} \tag{2.13}$$

$$T_{ml} = \frac{\overline{m} + \overline{l}}{2} \tag{2.14}$$

where $\overline{h}$, $\overline{m}$ and $\overline{l}$ are the average traffic density of training set's heavy, medium and light videos computed by the trained SVR model.

Table 2.7 contains the classification accuracy obtained by our method with features extracted from different layers and with 4 different SVR kernels. As can be seen, our global regression model gets very accurate results, all above 94.8%. The features from Conv-4 and Conv-5 are slightly better than those from FC-6 and FC-7. As for the Motorway dataset, the use of non-linear kernels also improve accuracy.

The estimated traffic density using Conv-5 and HIK kernel (the top performing combination according to Table 2.7) of all 254 videos is plot in Fig.2.7. Three sections step out from that curve : Heavy traffic (1 to 44), Medium traffic (45 to 89), and Light traffic (90 to 254).

Table 2.8 shows results obtained by other methods [20, 129, 32, 121, 6, 45, 99] all relying on motion or dynamic texture features (the numbers come from the original papers). As can be seen, the best state-of-the-art method [45] using a complex dynamic texture model reaches an accuracy of 97.23% on this dataset. Our results

Table 2.7 – Classification accuracy on UCSD dataset of using different layers' features and 4 different kernels SVR

	Conv-4	Conv-5	FC-6	FC-7
Linear	95.67	94.88	95.67	95.28
Hellinger	96.46	96.85	97.24	96.06
Chi2	96.85	97.24	96.06	96.46
HIK	97.24	97.64	96.46	96.85

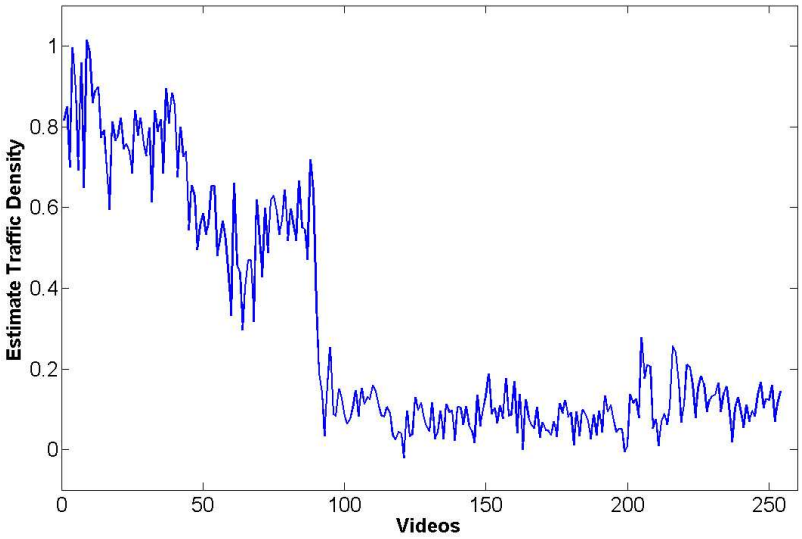

Figure 2.7 – The estimated traffic density of UCSD traffic dataset's 254 videos. The density curve shows 3 different sections which are correlated to the ground truth (Heavy: 1-44, Medium: 45-89, and Light: 90-254)

.

obtain by Conv-4 with HIK kernel has the same accuracy as [45] and using Conv-5 layer's feature can still improve the accuracy. We also find that FC-6 and FC-7 layer's features work slightly worse than Conv-4 and Conv-5 layer's feature, but still can outperform most of previous motion-based methods. This clearly shows that traffic density is strongly correlated to its 2D texture distribution and that motion features are not an explicit requirement for estimating traffic density.

Table 2.8 – Results of various methods on the UCSD dataset

Method	Accuracy	Method	Accuracy
Chan[20]	94.50%	Caffe[99]	96.85%
Sobral[129]	94.50%	BOVW[99]	96.85%
Derpanis[32]	95.28%	Conv-4(HIK)	**97.23%**
Riaz[121]	95.28%	Conv-5(HIK)	**97.64%**
Asmaa[6]	95.28%	FC-6(Linear)	96.46%
Gonçalves[45]	**97.23%**	FC-7(Chi2)	95.67%

Figure 2.8 – Three representative frames of testing videos with different illumination condition, shooting angle and low frame-rate

2.6 Transfer Learning Experimental Results

In order to analyze the performance of the proposed transfer learning framework on SegCNN and RegCNN, traffic videos from ChangeDetection.net [46] have been used for evaluation. The main advantage of these videos is that they come with a pixel-wise accurate groundtruth which allows precise traffic measurements. Three representative frames of the testing videos are shown in Fig.2.8. The Highway video sequence has a 25 fps frame-rate and shares similar visual properties with the images of the Motorway dataset. However, the TunnelExit and Turnpike sequences have a very low frame-rate and show different illumination conditions and different camera viewpoint than in the Motorway dataset.

In Fig. 2.11, we plot the segmentation results obtained by the SegCNN model before and after transfer learning. As one can see, SegCNN model without transfer learning gets accurate results on the Highway video, but gets poor results on the Turnpike and TunnelExit videos, mainly because of the camera viewpoint and illumination difference. In fact, the SegCNN model without transfer learning does not

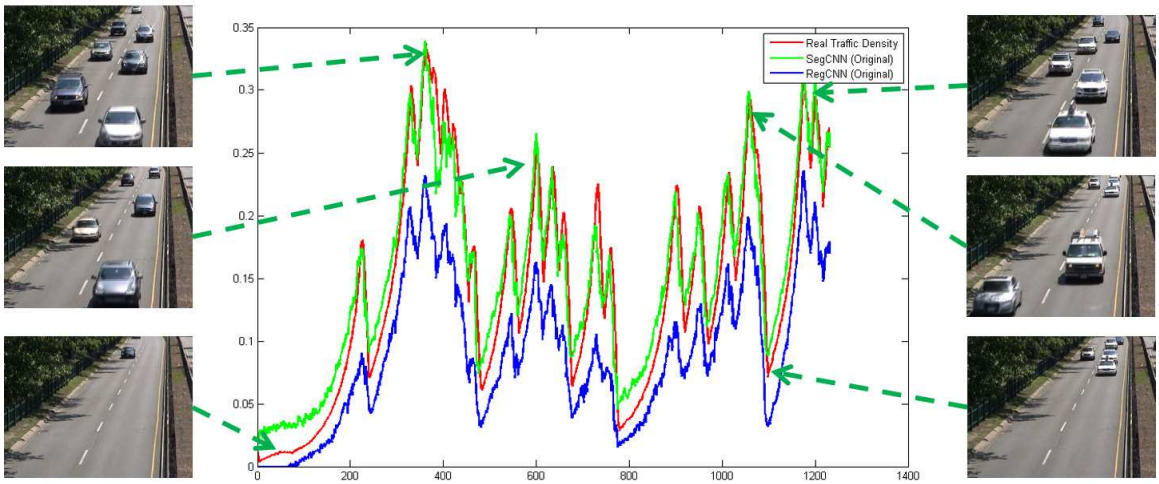

Figure 2.9 – Traffic density curves obtained by the original SegCNN and RegCNN on 1231 testing frames from the highway video

perform well on the 'Vehicle' and the 'Background' categories. However, after the transfer learning procedure, our SegCNN model produces far more accurate results, especially for the vehicles. Note that the proposed transfer model has a tendency of dilating the vehicle borders (c.f. Fig. 2.11). To deal with this issue, we erode the vehicle's probability map by 2 pixels and then update the RegCNN and Global model's parameter based on these eroded results.

In Fig. 2.9 we plot the estimated density curves of the original SegCNN and RegCNN on 1231 testing frames from the highway video, and in Fig.2.10 we also plot the density curves of the after transfer learned SegCNN and RegCNN. As can be seen, the output after using our transfer learning method (the blue and green curves in Fig.2.10) fit much more precisely on the real traffic density (the red curve) than the original models in Fig. 2.9.

In addition to the MAE, we also report the Pearson correlation coefficient to evaluate the global correlation between the true density and the estimated density throughout the videos:

$$Pearson_{X,Y} = \frac{Cov(X,Y)}{\sigma_X \cdot \sigma_Y} \tag{2.15}$$

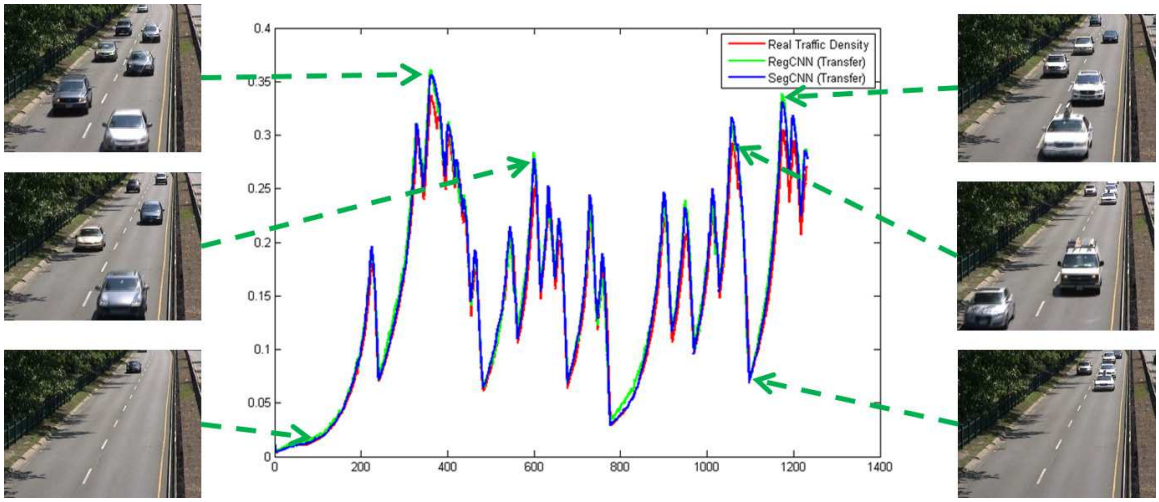

Figure 2.10 – Traffic density curves of the SegCNN and RegCNN after using the proposed transfer learning method on 1231 testing frames from the highway video

where X and Y are temporal density sequences like the curves in Fig. 2.9 and Fig. 2.10. The Pearson correlation value ranges between -1 and 1; it equals to 1 when X and Y are perfectly linearly correlated and decreases when the correlation between X and Y declines.

We tested the original SegCNN (which we refer to as SegCNN(O)), SegCNN with transfer learning (SegCNN(T)), the original RegCNN (RegCNN(O)), RegCNN with transfer learning (RegCNN(T)) and the Global model with transfer learning Global(T).

As can be seen from Table 2.9, for the highway and the Turnpike videos, the SegCNN(T) and RegCNN(T) which are both using the proposed transfer learning method have a better MEA and a better Pearson coefficient than SegCNN(O) and RegCNN(O). This is especially true for the Turnpike video where the SegCNN(O) model wrongly classified almost every vehicle. As a consequence, the estimated results are not correlated to the real traffic density, hence why the negative correlation. After the proposed transfer learning procedure, the SegCNN(T) model corrected most of these wrong classifications. Since RegCNN(T) and Global(T) are trained based on the results of SegCNN(T), we find that RegCNN(T) has similar result than SegCNN(T),

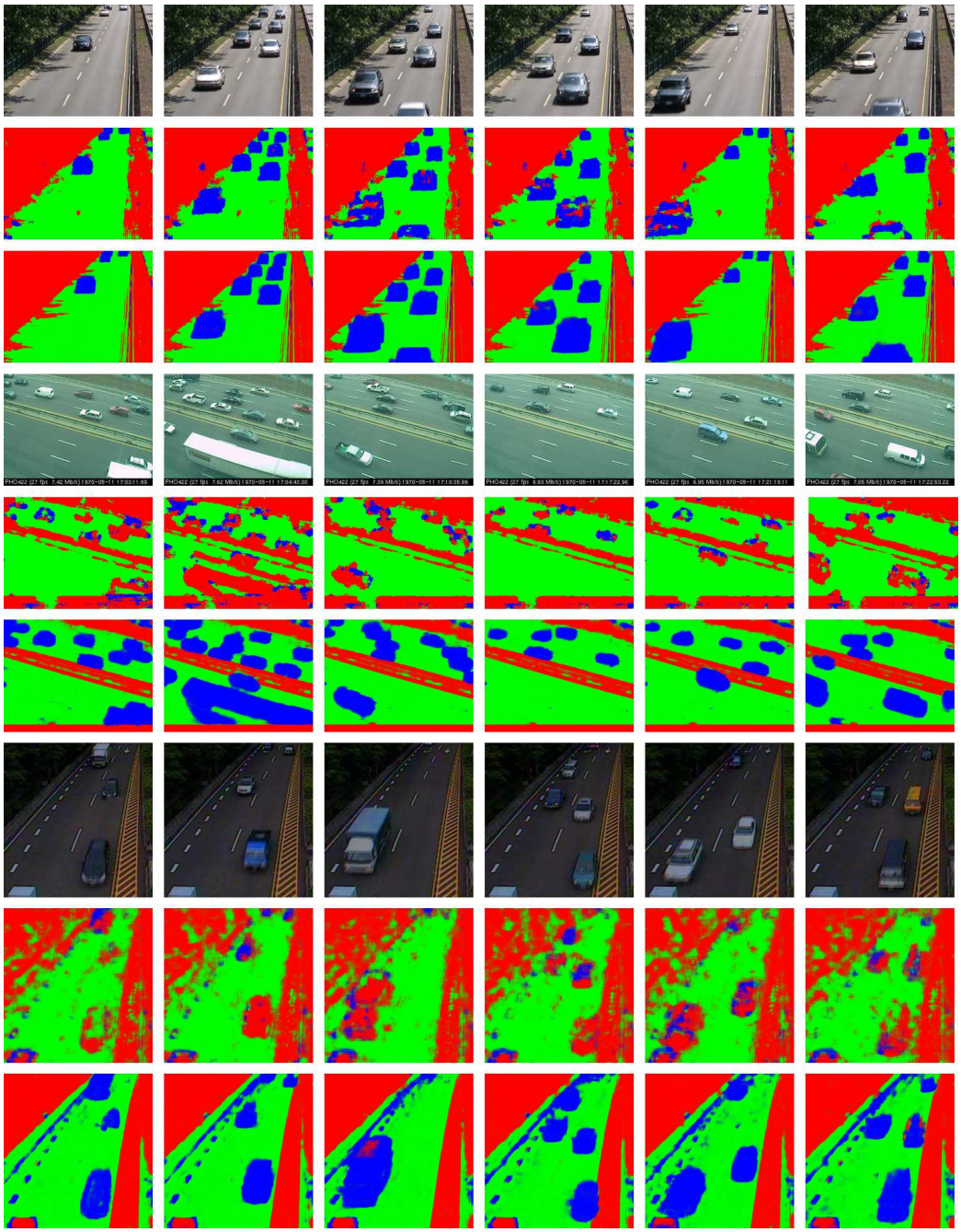

Figure 2.11 – The segmentation results on 3 testing videos. The middle rows show the result of the original SegCNN model, and the bottom rows are the results obtain by the SegCNN model after our transfer learning procedures.

while Global(T) is slightly less accurate.

As for the TunnelExit video, transfer learning was a deal breaker for RegCNN whose Pearson coefficient was critically low and the MAE abnormally high. However, we notice that the illumination variation gave rise to a series false detection of vehicles alongside of the road(c.f. the blue spots in the last row of Fig. 2.11). Those false detection appear systematically in every frame, and we measured that each frame void of any vehicle has an average 4184 false positive pixels. In order to gauge the influence of those erroneous pixels, we subtracted the numerator in our traffic density equation (Eq. 2.1) by 4184 and recompute each frame's density as follow: $Density = \frac{Vehicle-4184}{Vehicle+Road}$. By doing so, the MAE for SegCNN(T) decreases to 0.74, and the Pearson correlation is still 0.979. This implies that the traffic density curve is shifted by those erroneous pixels, hence why the MAE is large for SegCNN(T) (6.26) but at the same time its correlation to the ground truth is very good. Also, since RegCNN(T) is trained based on the segmentation results of SegCNN(T), it's normal that RegCNN(T) has a similar MAE and Pearson correlation value than SegCNN(T) as showed in Table. 2.9. This means that the estimated traffic density is much more accurate after transfer learning.

We also applied our transfer learning procedure to the FCN [97] model. As can be seen in Table. 2.9, on the Highway and Turnpike videos, the accuracy got improved, while on the TunnelExit video, we get a very slight increase on the Pearson coefficient. This shows that our transfer learning procedure accommodates to other CNN models, not only ours.

Three main conclusions shall be drawn from Table 2.9. First, a CNN trained on a traffic dataset showing similar road perspective and global illumination to the testing videos (here highway) can provide accurate results, whether it uses a global or a local model, with or without transfer learning. Second, when the camera orientation or the illumination varies significantly, transfer learning helps keeping a Pearson coefficient above 0.93, which is a satisfactory performance. Third, since all these results have been obtained after processing each frame independently, we see again that traffic can be accurately measured without the need of optical flow or object tracking, as is the case for other methods.

Table 2.9 – Quantitative evaluation with and without transfer learning

		MAE	Pearson			MAE	Pearson
Highway	SegCNN(O)	1.95	0.973	TurnPike	SegCNN(O)	8.15	-0.118
	SegCNN(T)	0.89	0.997		SegCNN(T)	2.72	0.987
	RegCNN(O)	5.40	0.965		RegCNN(O)	6.22	0.747
	RegCNN(T)	1.15	0.994		RegCNN(T)	3.25	0.981
	Global(T)	1.52	0.978		Global(T)	3.33	0.934
	FCN(O)	3.80	0.980		FCN(O)	6.66	0.815
	FCN(T)	1.44	0.981		FCN(T)	1.90	0.970
TunnelExit	SegCNN(O)	12.37	0.831				
	SegCNN(T)	6.26	0.979				
	RegCNN(O)	1.79	0.838				
	RegCNN(T)	6.31	0.961				
	Global(T)	4.96	0.933				
	FCN(O)	2.18	0.799				
	FCN(T)	2.26	0.851				

2.7 Conclusion

In this paper, we showed that highway traffic can be assessed using convolutional neural networks (CNN). In order to estimate traffic density, we proposed 3 different CNN models. The first one is a global regression model which uses a pre-trained ImageNet CNN to extract features of traffic images and a SVR to predict the traffic density. The second and third models are local patch based CNN models. The second model (SegCNN) turns the traffic density problem into a segmentation problem by classifying each pixel into 3 different categories (Vehicle, Road and Background) based on the context of the local patch. The third model (RegCNN) computes the traffic density by combining the predictions over multiple overlapping local image patches of Road and Vehicle pixel ratios.

Furthermore, we proposed a two-step transfer learning framework to adjust a CNN model's parameters for analyzing traffic surveillance video data which are captured in different illumination condition and different viewpoints.

Chapitre 3

MIO-TCD : A new benchmark dataset for vehicle classification and localization

Résumé

We introduced a new benchmark dataset (MIO-TCD) for vehicle localization and classification, which is the largest traffic surveillance dataset release to public. Based on this dataset, we organized the Traffic Surveillance Workshop and Challenge in conjunction with IEEE Conference on Computer Vision and Pattern Recognition (CVPR) 2017. In this chapter, we evaluated several different state-of-the-art deep learning methods for the classification and localization task on the MIO-TCD dataset, and identify scenarios for which these methods are still failing and propose concrete ideas for future work.

This chapter has been submitted to the journal IEEE Transactions on Image Processing.

Commentaires

The MIO-TCD dataset was built by Andrew Achkar, Akshaya Mishra, Justin Eichel, Pierre-Marc Jodoin, the Ph.D candidate and colleagues in

the Miovision Inc.. The evaluation website for the dataset was made and maintained by the Ph.D candidate. Experiments for the classification part were run by Carl Lemaire and Frederic B.-Charron, and experiments for the localization part were run by the Ph.D. candidate and Frederic.

The Ph.D. candidate, Carl Lemaire, Frederic B.-Charron and Pierre-Marc Jodoin wrote the paper. Janusz Konrad, Askhaya Mishra, Andrew Achkar, Justin Eichel and Shao-Zi Li helped for revising the paper.

MIO-TCD: A new benchmark dataset for vehicle classification and localization

Zhiming Luo
School of Information Science and Technology, Xiamen University,
Fujian Key Laboratory of Brain-like Intelligent Systems
Xiamen, Fujian, 361005 China
Département d'informatique, Université de Sherbrooke,
Sherbrooke, Québec, Canada J1K 2R1
zhiming.luo@usherbrooke.ca

Frederic B.-Charron
Département d'informatique, Université de Sherbrooke,
Sherbrooke, Québec, Canada J1K 2R1
frederic.branchaud-charron@usherbrooke.ca

Carl Lemaire
Département d'informatique, Université de Sherbrooke,
Sherbrooke, Québec, Canada J1K 2R1
carl.lemaire@usherbrooke.ca

Janusz Konrad
Department of Electrical and Computer Engineering, Boston University,
Boston, USA
jkonrad@bu.edu

Shao-Zi Li
School of Information Science and Technology, Xiamen University,
Fujian Key Laboratory of Brain-like Intelligent Systems
Xiamen, Fujian, 361005 China
szlig@xmu.edu.cn

Akshaya Mishra
Systems Design Engineering Department, University of Waterloo,
Waterloo, Canada
akshaya.ku.mishra@gmail.com

Andrew Achkar
Miovision Technologies Inc.
Kitchener, Canada
aachkar@miovision.com

Justin Eichel
Miovision Technologies Inc.

Kitchener, Canada
jeichel@miovision.com
Pierre-Marc Jodoin
Département d'informatique, Université de Sherbrooke,
Sherbrooke, Québec, Canada J1K 2R1
pierre-marc.jodoin@usherbrooke.ca

Keywords: Traffic analysis, deep learning, vehicle localization, vehicle classification.

Abstract

In this paper, we demonstrate the viability of deep learning for vehicle localization and classification tasks in traffic surveillance. We focus on the localization and classification of 11 different object classes such as cars, bicycles, pedestrians, and articulated trucks. Large datasets are a key aspect of deep learning. Here we introduce the "MIOvision Traffic Camera Dataset" (MIO-TCD), the largest dataset ever made for motorized traffic analysis. The dataset contains 786,702 annotated images acquired at different times of the day and different periods of the year by nearly a thousand traffic cameras deployed across Canada and the United States. The dataset consists of two parts: a "localization dataset", containing 137,743 full video frames with bounding boxes around traffic objects, and a "classification dataset", containing 648,959 crops of traffic objects from 11 classes.

In the wake of the 2017 CVPR MIO-TCD challenge, we show how well-trained, state-of-the-art deep learning methods can reach an accuracy and a Kappa score of more than 96% on the classification dataset and a mean-average precision of 77% on the localization dataset. We also identify scenarios for which state-of-the-art methods are still failing and propose concrete ideas for future work. Both the dataset and detailed results are publicly available on-line [102].

3.1 Introduction

The localization and classification of vehicles in the field of view of a traffic camera is the very first step of most traffic surveillance systems (e.g., car counting, activity recognition, anomaly detection, tracking, post-event forensics). Although subsequent processing may be different in each case, one often has to start by localizing foreground objects and then identifying what these objects are (trucks, cars, bicycles, pedestrians, etc.).

To our knowledge, except for pedestrian applications, most traffic monitoring systems rely on motion features such as optical flow, motion detection, and vehicle tracking [149, 155, 93]. Motion features are then used to count the number of vehicles on the road, estimate traffic speed, and recover global motion trajectories. But these traffic monitoring systems all share a fundamental limitation: in order to function properly, they need high frame-rate videos so motion features can be reliably extracted [72]. Also, reliable object tracking is still a challenge for cluttered scenes, especially when vehicles are partly occluded [98].

Unfortunately, low frame-rate videos are common as many large-scale camera networks cannot stream and store high frame-rate videos gathered by thousands of cameras. Instead, cameras are often configured to send one frame every second or so to the server. This is especially true for cameras transmitting over a cellular network whose bandwidth may vary over time making the throughput unpredictable. In such cases, one can only analyze ultra low frame-rate videos out of which no motion features can be accurately extracted. Also, those cameras have low resolution which makes localization and classification difficult.

To date, many object localization algorithms have been developed, and even more articles have been written on the topic. With the rise of deep learning methods (mostly convolutional neural nets), an exponential number of publications have been devoted to deep architectures implementing object recognition and localization methods [44, 43, 120, 55]. As a result, error rates on very challenging datasets such as Pascal VOC [37], ImageNet [123] and Microsoft COCO [91] have decreased at a steady pace. Among other things, what makes deep learning so successful is the availability of large and well-annotated datasets. Unfortunately, despite the large number of publications

devoted to traffic analytics, no such traffic dataset has been released to date. The lack of such a dataset has a number of implications, the most important one being that deep architectures trained on non-traffic oriented datasets such as ImageNet and COCO do not generalize well to traffic images.

Recognizing the importance of datasets to traffic analysis, we show how well-trained, state-of-the-art deep learning methods can be used to localize and identify moving objects with high precision without having to rely on motion features. This was made possible by leveraging a novel dataset called the MIOvision Traffic Camera Dataset (MIO-TCD). This dataset contains a total of 786,702 images: 137,743 high-resolution video frames with multiple vehicles in each frame and 648,959 vehicles cropped out of full frames. These images have been captured by nearly a thousand cameras deployed in urban and rural areas taking pictures at different periods of the year, at different times of the day, with different camera orientations and with various traffic densities. Nearly 200 people have participated in the process of hand-annotating each image to provide a bounding box around each clearly distinguishable object.

We believe that this work will have a substantial impact as the need for traffic monitoring systems working on still 2D images is becoming a glaring issue. With classification accuracies of up to 96% and localization mean-average precision of 77%, we show how one can localize and recognize vehicles regardless of their orientation, scale, color and environmental condition. As such, we expect our work will have a similar impact on the video surveillance community as the Caltech [33] and Inria [30] datasets had on the pedestrian detection community or ImageNet and COCO on the deep learning community.

3.2 Previous datasets

Nowadays, an exponentially large number of surveillance cameras are deployed along the roads of almost every country in the world. However, the images acquired by those cameras are the sole property of traffic management departments and are rarely released to the public. Consequently, few datasets have been made public for traffic analysis research. The most widely used datasets can be roughly divided into

3 categories, namely: (1) datasets of images taken by on-board cameras and mainly aimed at autonomous driving, (2) datasets of vehicle images taken by non-surveillance cameras and mainly oriented towards automatic recognition of high-resolution images from the Internet and (3) datasets of vehicle images taken by surveillance cameras. Here, we present a brief survey of existing vehicle detection and localization datasets.

On-board camera datasets

KITTI benchmark Dataset [42] [1]: This is a large dataset collected by an autonomous driving platform which addresses several real-world challenges, including: stereo vision calculation, optical flow estimation, visual odometry/SLAM, 3D object detection and 3D object tracking. This dataset consists of video frames captured by cars traveling both in rural areas and on highways.

Cityscapes Dataset [26] [2]: This dataset focuses on the semantic segmentation of urban street scenes. It comes with 5,000 fully-annotated images and 20,000 weekly-annotated images. The annotated objects span 30 classes including eight different types of vehicles. Unfortunately, this dataset is limited to images acquired from urban areas (50 cities) and during summer daytime.

Tsinghua-Tencent Traffic-Sign Dataset [163] [3]: This is a large traffic-sign dataset containing 100,000 images with 30,000 traffic-sign instances. Images in this dataset cover various illumination and weather conditions but do not contain any labeled vehicles.

Vehicles captured by non-surveillance cameras

Stanford Car Dataset [79] [4]: This dataset contains 16,185 high-resolution images of 196 classes of cars. The vehicle classes include the brand, the model, and the year (e.g. 2012 Tesla Model S or 2012 BMW M3 coupe). This dataset is divided into 8,144 training images and 8,041 testing images. In addition to the large variety of vehicles, all pictures have excellent resolution and have been captured in good lighting

1. http://www.cvlibs.net/datasets/kitti/index.php
2. https://www.cityscapes-dataset.com/
3. http://cg.cs.tsinghua.edu.cn/traffic-sign/
4. http://ai.stanford.edu/ jkrause/cars/car_dataset.html

conditions. However, none of the pictures were taken in a top-down orientation which is usually the case with surveillance cameras.

Comprehensive Cars Dataset (web) [153] [5]: This is one of the largest car dataset currently available. It comes with a total of 136,727 images showing entire cars and 27,618 images showing car parts. The dataset comprises 1,716 vehicle models as well as five different attributes (maximum speed, displacement, number of doors, number of seats, and type of car). Unfortunately, this dataset has the same limitations as the Stanford Car Dataset as its images were taken in a context far different than that of a surveillance camera (arbitrary orientation and filming 24/7).

Traffic datasets from surveillance cameras

Comprehensive Cars Dataset (surveillance) [153]: This dataset contains 44,481 images of cars captured by frontal-view surveillance cameras. The ground truth provides the model and the color of each vehicle. The main limitation of this dataset comes from the frontal view of the pictures which makes it hard to generalize to an arbitrary camera orientation. Furthermore, this dataset focuses on cars, mini vans and pickup trucks, and does not contain any large articulated trucks, buses, motorcycles and pedestrians.

BIT-Vehicle Dataset [35] [6]: This dataset consists of 9,850 vehicle images. This is one of the most realistic datasets with images coming from real surveillance cameras. Vehicles are divided into six categories, namely bus, micro-bus, minivan, sedan, SUV and truck. Unfortunately, those images were all taken in a top-frontal view during daytime and clear weather, thus limiting the diversity needed for general traffic analysis applications.

Traffic and Congestions (TRANCOS) dataset [48] [7]: This dataset is used to count the number of vehicles on highly-congested highways. It consists of 1,244 images with 46,796 annotated vehicles, most of which are partially occluded. Images were captured by publicly available video surveillance cameras of the Dirección General de Tráfico of Spain. Unfortunately, no vehicle type is provided.

5. http://mmlab.ie.cuhk.edu.hk/datasets/comp_cars/index.html
6. http://iitlab.bit.edu.cn/mcislab/vehicledb/
7. http://agamenon.tsc.uah.es/Personales/rlopez/data/trancos/

GRAM Road-Traffic Monitoring (GRAM-RTM) Dataset [47] [8]: This is a benchmark dataset for multi-vehicle tracking. It consists of video sequences recorded by surveillance cameras under different conditions and with different platforms. Every vehicle has been manually annotated into different categories (*car, truck, van*, and big-truck). Each video contains around 240 different objects.

Figure 3.1 – Sample images from the 11 categories of the classification dataset.

As can be seen, several publicly-available vehicle datasets contain images that do not come from surveillance cameras. The KITTI benchmark dataset, the Cityscapes Dataset and the Tsinghua-Tencent Traffic-Sign Dataset contain images captured by on-board cameras that can hardly be used to train traffic surveillance methods. The Stanford Car Dataset and the Comprehensive Car Dataset (web) contain high-resolution pictures of vehicles mainly taken in frontal and size view and rarely in top-down view as is usually the case with traffic surveillance applications. Images from these datasets are usually applied to fine-grained vehicle analysis (counting the number of doors, identifying the vehicle brand and year, etc.). As for the Comprehensive Cars Dataset (surveillance), although it is one of the largest publicly-available surveillance dataset, its images come from a well-aligned frontal-view camera and show only one vehicle per image. It also shows images in daylight and good weather. Also, none of the datasets contain more than a couple of thousand images.

As for the BIT-Vehicle dataset, it has up to two vehicles per frame but contains less than 10,000 images. The TRANCOS dataset contains traffic jam images in which

8. http://agamenon.tsc.uah.es/Personales/rlopez/data/rtm/

vehicles are too small to be categorized while the GRAM-RTM dataset has only three video sequences.

3.3 MIO-TCD : Our Proposed Dataset

3.3.1 Dataset Overview

The MIO-TCD dataset contains a total of 786,702 images, 137,743 being high-resolution video frames showing multiple vehicles and 648,959 lower-resolution images of cropped vehicles. These images were acquired at different times of the day and different periods of the year by nearly a thousand traffic cameras deployed across Canada and the United States. Those images have been selected to cover a wide range of challenges and are typical of visual data captured in urban and rural traffic scenarios. The dataset includes images: (1) taken at different times of the day, (2) with various levels of traffic density and vehicle occlusion, (3) showing small moving objects due to low resolution and/or perspective, (4) showing vehicles with different orientations, (5) taken under challenging weather conditions, and (6) exhibiting strong compression artifacts. This dataset has been carefully annotated by a team of nearly 200 people to enable a quantitative comparison and ranking of various algorithms.

The MIO-TCD dataset aims to provide a rigorous facility for measuring how far state-of-the-art deep learning methods can go at classifying and localizing vehicles recorded by traffic cameras. The dataset consists of two components: the classification dataset and the localization dataset. The classification dataset is used to train and test classification algorithms whose goal is to predict the kind of vehicle located in a low-resolution image patch. The localization dataset may be used to train and test algorithms whose goal is to localize and recognize vehicles located in the image.

MIO-TCD - Classification Dataset

The classification dataset contains 648,959 low-resolution images divided into 11 categories: Articulated Truck, Bicycle, Bus, Car, Motorcycle, Non-Motorized Vehicle, Pedestrian, Pickup Truck, Single-Unit Truck, Work Van and Background. As shown in Figure 3.1, each image contains a predominant object located in the middle of the

image. The objects pictured in that dataset come in various sizes and were recorded at various periods of the year, different times of the day and from different viewing angles. The dataset was split into 80% training (519,164 images) and 20% testing (129,795 images). Note that the car category contains vehicles of type sedan, SUV and family van.

Figure 3.2 – Sample images from the MIO-TCD localization dataset.

The number of images in each category is listed in Table 3.1. Since these images come from real footage, the number of samples per category is highly imbalanced as certain types of vehicle are more frequent than others. As such, almost half the images in the dataset fall into the car category, while the Bicycle, Motorcycle and Non-Motorized Vehicle contains roughly 2,000 training images and 500 testing images. We also randomly sampled 200,000 images to construct a background category.

MIO-TCD - Localization Dataset

The localization dataset contains 137,743 images of which 110,000 are intended for training and 27,743 – for testing. These images have various resolutions, ranging from 720×480 to 342×228. A total of 416,277 moving objects have been manually annotated with a bounding box and a category label. Except for Background, the same category labels as those in Table 3.1 have been used. However, due to the fact that localizing and recognizing vehicles in a full video frame is more difficult than classifying images of already localized vehicles, we added a new Motorized Vehicle

Table 3.1 – Size of each category in the classification dataset

Category	**Training**	**Testing**
Articulated Truck	10,346	2,587
Bicycle	2,284	571
Bus	10,316	2,579
Car	260,518	65,131
Motorcycle	1,982	495
Non-Motorized Vehicle	1,751	438
Pedestrian	6,262	1,565
Pickup Truck	50,906	12,727
Single-Unit Truck	5,120	1,280
Work Van	9,679	2,422
Background	160,000	40,000
Total	**519,164**	**129,795**

category which is unique to the localization dataset. This category contains all vehicles that are too small (or occluded) to be labeled into a specific category. Because of this category overlap, the classification dataset can be leveraged to improve the accuracy of localization methods.

Similarly to the classification dataset, images of the localization dataset came from nearly a thousand real traffic surveillance cameras. The category labels are thus imbalanced in similar proportions as those in Table 3.1 (see Figure 3.2 for some examples).

3.3.2 Evaluation Metrics

Classification

Three metrics have been implemented to gauge performance of classification methods. The first metric is the overall accuracy (Acc) which is the proportion of correctly classified images in the whole dataset:

$$Acc = \frac{TP}{\text{total number of images}} \tag{3.1}$$

where TP is the total number of correctly-classified images regardless of their category. TP can also be seen as the trace of a confusion matrix such as the one in Figure 3.3.

Since the classification dataset is highly imbalanced, large categories such as Car and Background have an overwhelming influence on the calculation of the overall accuracy. We thus implemented three metrics which account for this imbalance, namely the mean recall (mRe), the mean precision (mPr) and the Cohen Kappa Score ($Kappa$) [65].

The mean recall and mean precision are obtained by averaging the recall and the precision of each category thus giving an equal weight to each category. This is done as follows:

$$mRe = \frac{\sum_{i=1}^{11} Re_i}{11} \qquad mPr = \frac{\sum_{i=1}^{11} Pr_i}{11}$$

where $Re_i = TP_i/(TP_i+FN_i)$ and $Pr_i = TP_i/(TP_i+FP_i)$ are the recall and precision for category i, and TP_i, FN_i, FP_i are the number of true positives, false negatives and false positives for the i-th category, respectively.

$Kappa$ is a measure that expresses the agreement between two annotators. In our case, the first annotator is a method under evaluation and the second annotator is the ground truth. The Cohen Kappa Score is defined as [65]:

$$Kappa = \frac{Acc - P_e}{1 - P_e} \tag{3.2}$$

where Acc is the accuracy (3.1) and P_e is the probability of agreement when both annotators assign random labels. $Kappa$ values are in the $[-1, 1]$ range where $Kappa = 1$ means that both annotators are in complete agreement, while $Kappa \leq 0$ means no agreement at all.

Localization

Following the Pascal VOC 2012 object detection evaluation protocol, we report localization results via precision/recall curves and use the average precision (AP) as the principal quantitative measure for each vehicle category. A detection is considered a true positive when the overlap ratio between the predicted bounding box and

the ground-truth bounding box exceeds 50%; otherwise, it is considered a false positive. We also report the mean-average precision (mAP) which is the mean of AP across all categories. We invite the reader to refer to the Pascal VOC development kit document [9] for more details on the AP metric.

The main difference between our localization challenge and other localization challenges is the occurrence of the Motorized Vehicle category. This category is used to label vehicles that are too small (or too occluded) to be correctly assigned a specific category. For example, a car seen from far away and whose size does not exceed a few pixels would be labeled as a Motorized Vehicle. In order not to penalize methods which would incorrectly label those small objects, every Motorized Vehicle labeled by one of following categories: (Articulated Truck, Bus, Car, Pickup Truck, Single-Unit Truck and Work Van), is considered a true detection. In our implementation, we first identify every predicted bounding box whose overlap ratio with a Motorized Vehicle's ground-truth bounding box exceeds 50% , and re-assign its category label to Motorized Vehicle. Then, we compute the AP for each category.

3.4 Methods Tested

One of the overarching objectives of this paper is to identify how far state-of-the-art machine learning methods can go at classifying and localizing vehicles pictured by real traffic surveillance cameras. As mentioned earlier, this implies the analysis of low resolution 2D images with strong compression artifacts, recorded during daytime/nighttime, in different seasons, under diverse weather conditions, and with various camera positions and orientations. As such, one can expect that some methods might be robust to those challenges while others may not. In order to identify solved and unsolved issues, we carefully benchmarked a series of state-of-the-art deep learning methods. In this section, we describe the methods that we tested on the MIO-TCD classification and localization datasets. Some methods have been published before the release of the MIO-TCD dataset while others have been designed specifically for this dataset's challenges [102].

9. http://host.robots.ox.ac.uk/pascal/VOC/voc2012/devkit_doc.pdf

3.4.1 Classification

Pre-Trained CNN Features + SVM

The first series of methods aim at gauging how “different" is the statistical content of traffic images in the MIO-TCD dataset from other datasets, such as ImageNet. We did so by training a linear SVM classifier on CNN features obtained from ImageNet pre-trained CNN models. This is inspired by Razavian et al. [124] who demonstrated that features obtained from deep CNN models could be the primary choice of a large variety of visual recognition tasks. We did so with the following six pre-trained deep models: AlexNet [80], InceptionV3 [132], ResNet-50 [55], VGG-19 [127], Xception [25] and DensetNet [62]. The layer we used to extract the features from is: ReLU of FC-7 in AlexNet and VGG-19, and the global pooling layer in InceptionV3, ResNet-50, Xception and DenseNet. Their corresponding feature dimensions are 4096, 4096, 2048, 2048, 2048, 1920. The python scikit-learn library [10] was used to train the linear SVM with $C = 1$.

Retrained CNN Models

Although features from the first layers of a CNN model are independent from the dataset it was trained on (mostly Gabor-like filters [154]), we retrained end-to-end all six CNN models on our classification dataset. Those models were all initialized with ImageNet pre-trained weights and were trained with the loss function proposed in the corresponding original papers. We used the Adam [76] optimizer with a learning rate of 10^{-3} that we empirically found more effective than other optimizers such as RMSprop or SGD. Training was done for a maximum of 50 epochs with a validation-based early stopping criteria with a patience of 10 epochs to prevent over-fitting. The batch size was adjusted to each model so it could fit on our 12GB Titan X GPUs. Please note that we included a series of batch normalization layers [63] to AlexNet and VGG-19 to speed up training. All models but DenseNet were implemented with the Keras library [24]. DenseNet was implemented with PyTorch [114].

We also implemented the following four training configurations to further improve results on our dataset:

10. scikit-learn.org/

— The first configuration is a basic training with normal sampling of the data.
— The second configuration involves data augmentation using horizontal flipping and randomized shearing and zooming to enrich the training dataset.
— Since the dataset is highly imbalanced (see Table 3.1) we used data augmentation with uniform sampling. That is, at each epoch we used an equal number of images from each class to prevent large classes, such as *Car*, *Pickup Truck* and *Background* from gaining too much importance over smaller classes, such as *Motorcycle*.
— As suggested by Havaei et al. [52], we implemented a two-phase training procedure. In the first phase, we use data augmentation with uniform sampling. In the second phase, we freeze the entire network except for the last layer which is retrained with data augmentation and normal sampling.

MIO-TCD Classification (Ensemble Models)

In the wake of the 2017 CVPR MIO-TCD Challenge [102], several methods have been designed for the sole purpose of classifying traffic images. Interestingly, all of those methods involve a combination of several deep learning models. Kim and Lim [75] proposed a bagging system where several CNN models are trained on random subset of the MIO-TCD dataset. Their final result is obtained with a weighted majority vote to compensate for the unbalanced nature of the dataset. Lee and Chung [85] proposed an ensemble method which combines 3 convolutional networks (AlexNet, GoogleNet and ResNet18) trained on 18 different sets of data. GoogleNet was trained on 12 subsets of the dataset (aka the local nets) and AlexNet, GoogleNet and ResNet18 were trained on the entire dataset but with different image sizes (aka the global nets). At test time, the networks are selected with a gating function and combined with a softmax layer. Jung et al. [70] proposed an ensemble model according to which several deep residual networks are jointly trained. The main novelty of their method lies within its loss function which allows to train every ResNet simultaneously. Theagarajan et al. [134] also proposed an ensemble of ResNet models. The authors implemented a weighted loss function to account for the unbalanced nature of the dataset. They also implemented a patch-wise logical reasoning process to disambiguate classes that are close to each other like trucks and buses.

3.4.2 Localization

Recently, CNN-based localization methods have established state-of-the-art performances on several object localization datasets. In this paper, we evaluated Faster R-CNN [120], SSD-300, SSD-512 [96], YOLO [119] and YOLO-v2 [118].

Faster R-CNN is an improved version of Girshich et al.'s R-CNN [44] and Fast R-CNN [43] methods. Unlike R-CNN and Fast R-CNN, that use a selective search [137] to generate object bounding box proposals, Faster R-CNN has a region proposal network that can directly estimate proposals based on the CNN feature maps thus making it end-to-end trainable. Liu *et al.* [96] improved the Faster R-CNN model with their SSD (Single Shot MultiBox Detector) framework which generates object proposals with feature maps from multiple layers. As for YOLO (You Only Look Once) [119], it takes a 448×448 image as input and outputs object detections within a 7×7 grid. Later on, Redmon *et al.* [118] integrated anchor boxes (for computing potential bounding boxes) into their YOLO model. We refer to this model as YOLO-v2.

We trained Faster R-CNN end-to-end for 300,000 iterations with the code provided by the authors [11]. As recommended by the original paper, we used VGG-16 as the pre-trained model. Similarly, for SSD-300 and SSD-512 (i.e., SSD with input images resized to 300×300 and 512×512, respectively) we used the code released by the authors [12]. The SSD-300 model was trained for 120,000 iterations with a batch-size of 32, and the SSD-512 model was trained for 240,000 iterations with a batch-size of 16. YOLO was trained using the Darknet deep learning toolbox [13]. Both YOLO and YOLO-v2 were trained for 80,000 iterations with a batch-size of 64. As YOLO-v2 needs pre-clustered anchor bounding boxes, we evaluated two kinds of anchor boxes: boxes computed on the Pascal VOC datasets (referred to as YOLO-v2(P)) and on our localization dataset (referred to as YOLO-v2(M)).

11. https://github.com/rbgirshick/py-faster-rcnn
12. https://github.com/weiliu89/caffe/tree/ssd
13. https://pjreddie.com/darknet/

MIO-TCD Localization

We report results from two localization methods designed specifically for the MIO-TCD dataset. The first one is from Wang et al. [147] which improves methods such as Faster R-CNN and SSD by leveraging the scene context. The idea is to combine the softmax score of these methods with a context term which the authors model with a k-NN algorithm. The second method is from Jung et al. [70] which combine the results computed from multiple R-FCN models [29] trained with different backbones (ResNet-50 and ResNet-101) together with a joint non-maximal suppression step to localize vehicles.

3.5 Experimental Results

In this section, we report experimental results obtained on the MIO-TCD classification and localization datasets with the deep learning methods presented earlier.

3.5.1 Classification

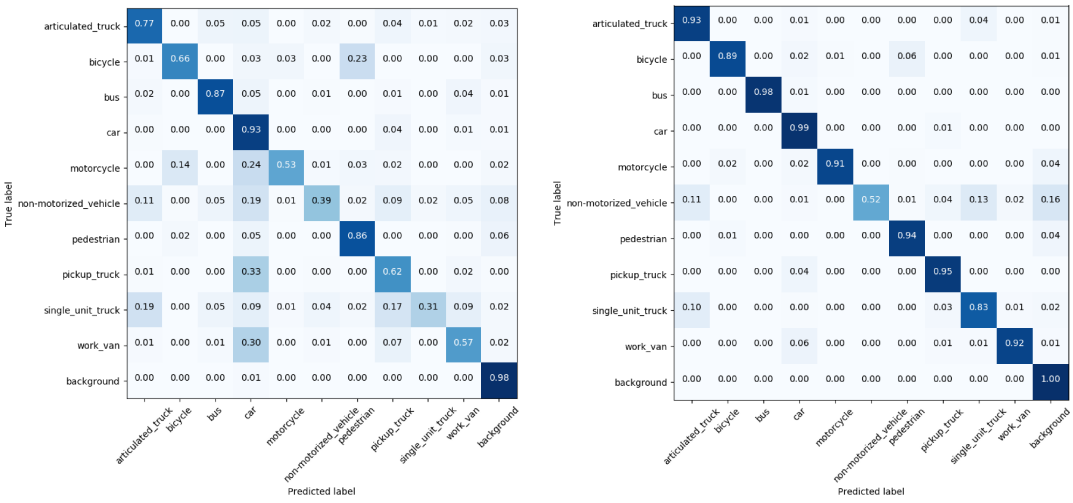

Figure 3.3 – Confusion matrices obtained on the classification dataset with pre-trained ResNet-50 features + SVM on left and the ensemble model by Jung *et al* [70] on right.

Table 3.2 – Evaluation metrics for six pre-trained models used with linear SVM classifiers on the classification dataset.

	Acc	*mRe*	*mPr*	*Kappa*
AlexNet	0.82	0.49	0.55	0.72
Inception-V3	0.84	0.57	0.64	0.75
ResNet-50	**0.89**	**0.69**	0.74	**0.83**
VGG19	0.83	0.66	0.59	0.75
Xception	0.87	0.54	0.76	0.78
DenseNet	0.86	0.51	**0.82**	0.78

Pre-Trained CNN Features + SVM

Results obtained with SVM trained on features extracted from six pre-trained CNN models are shown in Table 3.2. All six models have an accuracy of more than 80%, which is surprisingly high considering the different nature of the ImageNet and MIO-TCD datasets. However, a careful analysis reveals that these methods have relatively low mean recall, mean precision and Kappa score. To understand this situation, one has to consider the confusion matrix of ResNet-50 shown in Figure 3.3. While the *Car* and *Background* categories have an accuracy of more than 90%, others, such as *Non-Motorized Vehicle*, *Motorcycle* and the *Single-Unit Truck* categories, suffer from very low accuracy.

These numbers can be explained by the unbalanced nature of the dataset, as the *Car* and *Background* categories contain more than 80% of all images. In this case, large classes get to attract the decision boundaries on their side to the detriment of smaller classes. This also explains why several categories have a large proportion of samples wrongly classified as *Car*. It is the case for *Work Van* for which 30% of its images are classified as *Car*. A detailed analysis revealed that features trained on ImageNet do not discriminate family vans (labeled as *Car* in the dataset) from *Work Vans*. Confusion also happens between *Bicycle* and *Pedestrian*, as well as within the group of *Articulated Truck*, *Pickup Truck* and *Single-Unit Truck*.

Based on these results, we conclude that SVM trained on ImageNet CNN features is accurate at classifying large classes but this does not generalize well to smaller classes.

Retrained CNN Models

Table 3.3 reports evaluation metrics for the six models trained in four different training configurations. As can been seen, there is a substantial performance increase in comparison to the SVM results in Table 3.2. Results show that data augmentation improves the Kappa score of every model, especially ResNet-50. Note that although data augmentation may reduce the mean recall of certain method, that is compensated by a larger increase of the mean precision. However, the use of uniform sampling (+U) with and without the two-phase training procedure (T) does not improve results over the normal sampling (+N).

A careful inspection reveals that uniform sampling has a positive impact on small categories (such as Motorcycle, for example), but also decreases the performance for large categories.

Overall, Xception and DenseNet with data augmentation and normal sampling are the best methods with very similar performance. VGG-19, ResNet-50 and Inception-V3 also get very accurate results with Kappa scores above 0.93.

MIO-TCD Classification (Ensemble Models)

Table 3.4 reports results submitted to the 2017 CVPR MIO-TCD Challenge. These are ensemble methods which combine the output of different models. As can be seen, all these methods have similar performance with accuracy of about 0.98 and Kappa score of almost 0.97. With these results being only marginally better than those obtained with Xception and DenseNet, we conclude that the combination of several models does not bring much on a dataset such as MIO-TCD.

Error Analysis

Results from Tables 3.3 and 3.4 reveal that despite large illumination variations between images, compression artifacts, arbitrary vehicle orientation, poor resolution and inter-class similarities, the classification of traffic vehicles seems almost solved. However, in-depth analysis of the top-performing methods reveals some unsolved issues. In Figure 3.3, we show the confusion matrix for the method by Jung *et al.* [70] whose performance is globally similar to that obtained by other top performing methods. As

Table 3.3 – Evaluation metrics for retrained CNN models on the classification dataset in four different configurations. 'N' stands for normal sampling, 'U' stands for uniform sampling, 'D' means using data augmentation and 'T' is the two-phase training procedure. (Res-50 stands for ResNet-50 model, Incept. stands for Inception-V3 model.)

	Acc				*mRe*				*mPr*				*Kappa*			
	N	D+N	D+U	T	N	D+N	D+U	T	N	D+N	D+U	T	N	D+N	D+U	T
AlexNet	0.87	0.89	0.82	0.78	0.63	0.56	0.83	0.35	0.55	0.69	0.63	0.50	0.80	0.83	0.74	0.63
Incept.	0.96	0.96	0.93	0.96	0.83	0.89	0.89	0.87	0.85	0.84	0.76	0.89	0.93	0.94	0.89	0.94
Res50	0.94	0.97	**0.96**	**0.97**	0.79	0.87	0.89	**0.88**	0.73	0.86	0.86	0.90	0.90	0.95	**0.94**	**0.95**
VGG-19	0.94	0.96	0.89	0.94	0.77	0.82	0.65	0.82	0.79	0.85	0.84	0.84	0.91	0.93	0.83	0.91
Xception	0.96	**0.98**	0.94	0.96	0.85	**0.90**	0.82	0.82	0.79	0.91	**0.90**	0.84	0.94	**0.96**	0.91	0.94
DenseNet	**0.97**	0.97	0.95	0.95	**0.89**	0.87	**0.90**	0.82	**0.89**	**0.92**	0.85	**0.91**	**0.95**	**0.96**	0.93	0.93

Table 3.4 – Ensemble models on the MIO-TCD classification dataset.

	Acc	*mRe*	*mPr*	*Kappa*
Kim and Lim [75]	0.9786	0.9041	0.9355	0.9666
Lee and chung [85]	0.9792	0.9024	0.9298	0.9675
Jung *et al* [70]	**0.9795**	0.8970	**0.9530**	**0.9681**
Theagarajan *et al* [134]	0.9780	**0.9190**	0.9439	0.9658

Figure 3.4 – Examples of failure cases from top-performing methods for every class where the yellow label (top) is the ground truth and the white label (bottom) is the predicted class.

one can see, *Non-Motorized Vehicles* are poorly handled. Images of *Non-Motorized Vehicles* in our dataset include a wide variety of trailers pulled by a vehicle, typically a car or a pickup truck. As shown in Figure 3.4 (second row, third column), *Non-Motorized Vehicles* are often wrongly classified as a single-unit or an articulated truck, as their shapes are very similar.

Without much surprise, methods also get confused between categories with similar visual characteristics, such as the Work Van class and the Car class (more specifically, family vans considered to belong to the Car class) or the Articulated Truck and the Single-Unit Truck. They also get confused by vehicles with unusual look. For example, in Figure 3.4, the blue car with a black top gets wrongly classified as a pickup truck and the pickup truck with a cap is wrongly classified as a car. Also, classes belonging to small objects such as *Pedestrian*, *Bicycle* and *Motorcycle* often suffer from heavy compression artifacts and are thus more likely to be mis-classified.

Table 3.5 – Average precision (AP) of localization for Faster R-CNN, SSD, YOLO and two methods submitted to MIO-TCD Challenge on localization. (A.Truck: Articulated Truck, Bike: Bicycle,Motor: Motorcycle, M.Vehicle: Motorized Vehicle, N.Vehicle: Non-Motorized Vehicle, Ped.: Pedestrian, Pickup: Pickup Truck, S.Truck: Single-Unit Truck, W.Van: Work Van)

	mAP	A.Truck	Bike	Bus	Car	Motor	M.Vehicle	N.Vehicle	Ped.	Pickup	S.Truck	W.Van
Faster R-CNN	70.0	85.9	78.4	95.2	82.6	81.1	52.8	37.3	31.3	89.0	62.5	73.6
SSD-300	74.0	90.6	78.3	95.7	91.5	78.9	51.4	55.2	37.3	90.7	69.0	75.0
SSD-512	**77.3**	**92.1**	**78.6**	**96.8**	**94.0**	**82.3**	**56.8**	**58.8**	**43.6**	**93.1**	**74.0**	**80.4**
YOLO-v1	62.7	82.7	70.0	91.6	77.2	71.4	44.4	20.7	18.1	85.6	58.3	69.3
YOLO-v2(P)	71.5	86.7	78.4	95.2	80.5	80.9	52.0	56.5	25.7	84.6	70.0	75.7
YOLO-v2(M)	71.8	88.3	**78.6**	95.1	81.4	81.4	51.7	56.6	25.0	86.5	69.2	76.4
Wang *et al.*[147]	77.2	91.6	79.9	96.8	**93.8**	83.6	56.4	58.2	42.6	**92.8**	73.8	79.6
Jung *et al.*[70]	**79.2**	**92.5**	**87.3**	**97.5**	89.7	**88.2**	**62.3**	**59.1**	**48.6**	92.3	**74.4**	**79.9**

3.5.2 Localization

The average precision of localization for Faster R-CNN, SSD-300, SSD-512, YOLO-v1, YOLO-v2(P) and YOLO-v2(M) methods is shown in Table 3.5. As can be seen, the SSD methods outperform both Faster R-CNN and YOLO, with SSD-512 being the best-performing method with a mean-average precision (mAP) of 77.3%. At the bottom of the table are methods by Wang *et al.* [147] and Jung *et al.* [70] submitted to the MIO-TCD Challenge which attain even better results with a mAP of 79.2%.

Results for SSD-300 and SSD-512 show that increasing input image resolution from 300×300 to 512×512 improves the mAP by 4%. Also, YOLO-v2 has the mAP 10% higher than YOLO-v1 thus showing that anchor boxes are useful features. Furthermore, results for YOLO-v2(P) and YOLO-v2(M) show that anchor boxes computed on our localization dataset marginally improve mAP over anchor boxes pre-computed from the Pascal VOC dataset.

Here again, the largest classes, namely *Car*, *Pickup Truck*, *Articulated Truck* and *Bus* get the best results with an average precision above 80% for almost every method, while *Motorized Vehicle*, *Non-Motorized Vehicle* and *Pedestrian* are the three categories with the lowest average precision. The main challenge with *Motorized Vehicle* and *Pedestrian* classes stems from the small size of vehicles that are likely to be confused with the *Bicycle* and *Motorcycle* categories. As for the *Non-Motorized Vehicle*, similarly to the classification dataset, it is often confused with the *Articulated Truck* and *Single-Unit Truck* classes.

In Figure 3.5, we plot some detection results for different methods. While most methods can accurately localize large and well-contrasted vehicles, we can see that Faster R-CNN is prone to false detections while YOLO-v1 and YOLO-v2 suffer from mis-detections of small objects (typically, pedestrians).

Detailed Analysis

We now thoroughly analyze the influence of object scale on the performance of localization methods as well as the nature of false detections. We do so via the Microsoft COCO's evaluation procedure [91] and the object detectors' protocol by Hoeim et al. [57].

Figure 3.5 – Detection examples on the localization dataset for Faster R-CNN, SSD-300, SSD-512, YOLO, YOLO-v2 (Pascal VOC) and YOLO-v2 (MIO-TCD). We only show detections with probability scores higher than 0.6.

Scale: Every object has been classified as belonging to one of 3 scales: small objects with bounding box area below 32^2, medium objects with the area between 32^2 and 96^2, and large objects with the area larger than 96^2. The average precision for each of these scales is reported in Table. 3.6. As can be seen, all methods in the table are ill-suited for detecting small objects, in our case *Pedestrian*, *Bicycle*, *Motorcycle* classes as well as vehicles seen from a distance (see Figure 3.2 for examples of small vehicles due to perspective effect). Furthermore, when increasing the overlap ratio for correct detections from 0.5 to 0.75, we find that the AP of Faster R-CNN and YOLO decreases by almost 30%, while for SSD it decreases by around 15%. This means that the bounding boxes estimated by the SSD method are tighter around the ground-truth bounding boxes.

Table 3.6 – Average precision of localization computed using Microsoft COCO's evaluation protocol.

	Average Precision				
	Overlap		Scale		
	0.5	0.75	small	medium	large
Faster R-CNN	70.0	38.5	14.3	40.2	55.1
SSD-300	74.3	57.1	21.5	53.3	69.0
SSD-512	**77.6**	**61.9**	**28.2**	**57.3**	**72.5**
YOLO-v1	62.6	34.0	11.3	32.4	52.7
YOLO-v2(P)	71.3	42.7	15.7	41.3	59.3
YOLO-v2(M)	71.8	43.0	16.0	41.7	61.3
Wang *et al.* [147]	77.4	**59.9**	**26.6**	**55.6**	60.7
Jung *et al.* [70]	**79.3**	58.8	26.5	54.9	**69.3**

False positives: To examine the nature of false positives, we follow the methodology of Hoiem et al. [57] according to which each prediction is either correct or wrongly classified into one of following errors :

— **Localization**: the predicted bounding box has a correct label but is misaligned with the ground-truth bounding box ($0.1 < \text{Overlap} < 0.5$).

— **Similarity**: the predicted bounding box has Overlap > 0.1 with a ground-truth bounding box but its predicted label is incorrect. However, the predicted label belongs to one of three similarity classes (groups of similar classes),

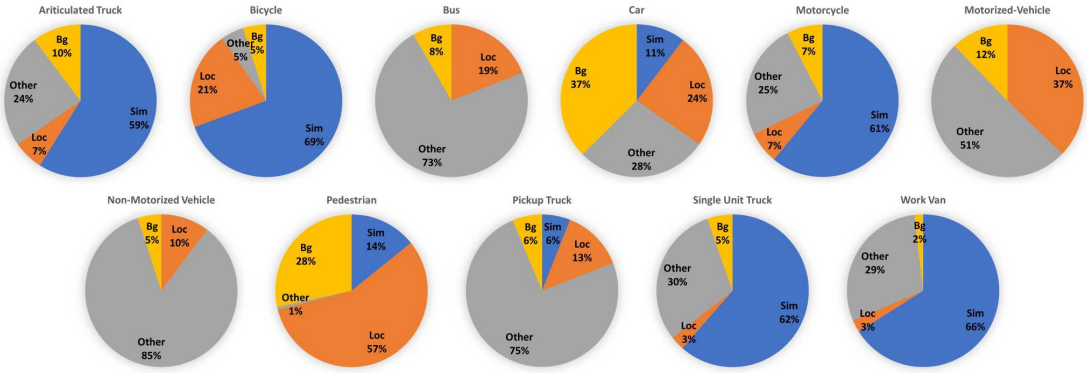

Figure 3.6 – Analysis of false detections by the SSD-300 method. Each pie-chart shows the fraction of top-ranked false positives of each category due to poor localization (Loc), confusion with similar categories (Sim), confusion with other categories (Other), or confusion with background or unlabeled objects (Bg).

namely : {*Articulated Truck, Pickup Truck, Single-Unit Truck*}, {*Bicycle, Motorcycle, Pedestrian*}, and {*Car, Work Van*}.

— **Other**: the predicted bounding box has a Overlap > 0.1 with a ground-truth bounding box but its predicted label is incorrect and does not fall within a group of similar classes.

— **Background**: all other false positives are classified as background, mainly confused with unlabeled objects.

Figure 3.6 shows the frequency of occurrence of each type of error for the SSD-300 method. For the similarity class {*Articulated Truck, Pickup Truck, Single-Unit Truck*}, the *Articulated Truck* and *Single-Unit Truck* classes are likely to be confused with each other, while the *Pickup Truck* is confused with other classes, mostly *Car*. As for the {*Car, Work Van*} similarity class, we find that 66% of *Work Van* false positives are wrongly classified as *Car* for the reason mentioned before (*Work Van* is often confused with a family van). As for *Car* false detections, the confusion is mostly with *Background*. For the {*Bicycle, Motorcycle, Pedestrian*} similarity class, the *Bicycle* and *Motorcycle* classes are often confused with each other, while *Pedestrian*, due to small scale, suffers from localization errors. As for the *Bus*, *Motorized Vehicle* and *Non-Motorized Vehicle* classes, they all suffer from confusion with other categories.

3.6 Conclusions

In this paper, we introduced the MIOvision Traffic Camera Dataset (MIO-TCD), the largest dataset ever made for motorized traffic analysis. The dataset consists of two parts: a "localization dataset", containing full video frames with bounding boxes around traffic objects, and a "classification dataset", containing crops of 11 types of traffic objects.

We evaluated several state-of-the-art deep learning methods on the MIO-TCD dataset. Results show that well-trained models reach impressive accuracy and Kappa scores of more than 96% on the classification dataset and a mean-average precision of 79% on the localization dataset.

While Xception and Densenet with data augmentation are the best classification methods, VGG-19, RestNet-50 and Inception-V3 attain very good results as well. As for the ensemble models, they reach, for all practical purposes, the same scores as Xception and Densenet. A careful inspection of results reveals that Non-Motorized Vehicles is the only problematic class with a precision below 80%. Other errors in the top results can be explained by the confusion between classes with similar visual characteristics such as Single-Unit Truck and Articulated Truck.

As for localization methods, the method by Jung et al. [70] gets the best scores (mAP=79.2%) but is closely followed by SSD-512 (mAP=77.3%). A detailed analysis reveals that errors often happen between similar classes or are due to a mis-alignment of the predicted bounding box (overlapping ratio below 0.5).

In light of these results, we may conclude that state-of-the-art deep learning methods exhibit a capacity to localize and recognize vehicles from single video frames without the need for dynamic features captured by video, as was required to date. This opens the door to new, low-frame-rate video analytics applications such as traffic statistics, traffic density estimation, car counting, and anomaly detection.

Chapitre 4

Non-Local Deep Features for Salient Object Detection

Résumé

We address the issue of saliency detection which aims to highlight the most relevant objects in an image (eg. vehicles in traffic scenes). We propose a simplified convolutional neural network which combines local and global information through a multi-resolution 4×5 grid structure, and the model is trained based on loss functon inspired by the Mumford-Shah (MS) functional [111] which penalizes error on the boundary. The model enables near real-time high performance saliency detection. We also extent this model to do traffic analysis (vehicle segmentation). A model is trained on the MIO-TCD dataset which can do accurate vehicle segmentation, and several traffic videos taken from the CDNet dataset [46] was used to test generalization ability.

This chapter was accepted for a spotlight presentation at the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) 2017.

Commentaires

The Ph.D. candidate proposed the initial idea and did all the experiments in the paper. The Ph.D canidate, Akshaya Mishra, Andrew Achkar and Pierre-Marc Jodoin wrote the paper together. Justin Eichel and Shao-Zi Li helped for revising the paper.

Non-Local Deep Features for Salient Object Detection

Zhiming Luo
School of Information Science and Technology, Xiamen University,
Fujian Key Laboratory of Brain-like Intelligent Systems
Xiamen, Fujian, 361005 China
Département d'informatique, Université de Sherbrooke,
Sherbrooke, Québec, Canada J1K 2R1
zhiming.luo@usherbrooke.ca

Akshaya Mishra
Miovision Technologies Inc.
Kitchener, Canada
amishra@miovision.com

Andrew Achkar
Miovision Technologies Inc.
Kitchener, Canada
aachkar@miovision.com

Justin Eichel
Miovision Technologies Inc.
Kitchener, Canada
jeichel@miovision.com

Shao-Zi Li
School of Information Science and Technology, Xiamen University,
Fujian Key Laboratory of Brain-like Intelligent Systems
Xiamen, Fujian, 361005 China
szlig@xmu.edu.cn

Pierre-Marc Jodoin
Département d'informatique, Université de Sherbrooke,
Sherbrooke, Québec, Canada J1K 2R1
pierre-marc.jodoin@usherbrooke.ca

Abstract

Saliency detection aims to highlight the most relevant objects in an image. Methods using conventional models struggle whenever salient objects are pictured on top of a cluttered background while deep neural nets suffer from

excess complexity and slow evaluation speeds. In this paper, we propose a simplified convolutional neural network which combines local and global information through a multi-resolution 4×5 grid structure. Instead of enforcing spacial coherence with a CRF or superpixels as is usually the case, we implemented a loss function inspired by the Mumford-Shah functional which penalizes errors on the boundary. We trained our model on the MSRA-B dataset, and tested it on six different saliency benchmark datasets. Results show that our method is on a par with the state-of-the-art while reducing computation time by a factor of 18 to 100 times, enabling near real-time, high performance saliency detection.

4.1 Introduction

Saliency detection aims to mimic the human visual system which naturally separates predominant objects of a scene from the rest of the image. Several applications benefit from saliency detection including image and video compression [50], context aware image re-targeting [87], scene parsing [157], image resizing [3], object detection [144] and segmentation [107].

A salient object is often defined as a region whose visual features differ from the rest of the image and whose shape follows some a prior criteria [13]. Traditional methods typically extract local pixel-wise or region-wise features and compare it with global features. The result of that comparison is called a saliency score which is stored in a saliency map. Recently, deep learning has entered the field of saliency detection and quickly established itself as the de facto benchmark. Their greatest asset relative to traditional unsupervised approaches is that they can be trained end-to-end using simple optimization functions that combine local and deep features.

While some methods apply a straight forward convolutional neural net (CNN) model [113], others have proposed a model tailored to the saliency detection problem [87, 88, 94, 143, 157]. To achieve state-of-the-art performance, the top performing CNN models require non-trivial steps such as generating object proposals, applying post-processing, enforcing smoothness through the use of superpixels or defining complex network architectures, all the while making predictions far slower than real-time.

As such, there remain opportunities to simplify the model architecture and speed up the computation.

In this paper, we show that the overarching objectives of state-of-the-art CNN models (enforcing spatial coherence of the predicted saliency map and using both the local and global features in the optimization) can be achieved with a much simplified non-local deep feature (NLDF) model. Spatial coherence is enforced with a Bayesian loss inspired by the Mumford-Shah (MS) functional [111]. The loss is expressed as the sum of a cross-entropy term and a boundary term. As opposed to conventional implementations of the MS functional, we use non-local features learned by a deep network instead of raw RGB colors. Also, rather than minimizing the boundary length directly (as done by unsupervised MS implementations), we minimize an intersection over union loss computed using predicted and ground truth boundary pixels. This boundary penalty term is shown to contribute significantly to our model's performance.

Our model's network is composed of convolution and deconvolution blocks organized in a 4×5 grid (see Figure 4.1) where each column of the grid extracts resolution-specific features. Local contrast processing blocks are also used along each resolution axis in order to promote features with strong local contrast. The resulting local and global features are combined into a "score" processing block that gives the final output at half of the input resolution.

Since our method does not rely on superpixels, it is fully convolutional and thus achieves best-in-class evaluation speeds. The NLDF model evaluates an input image in 0.08s, a speed gain of 18 to 100 times as compared to other state-of-the-art deep learning methods, while being on a par with state-of-the-art evaluation performance on the MSRA-B[95], HKU-IS[87], PASCAL-S[90], DUT-OMRON[152], ECSSD [151] and SOD [105] benchmark datasets.

The rest of the paper is organized as follows. Section 4.2 provides an overview of deep learning based saliency detection techniques. Section 4.3 describes the theory and practical implementation of our NLDF model. Finally, Section 4.4 discusses the performance of non-local feature model compared to other state-of-the-art saliency detection methods.

4.2 Related Works

Most previous methods implement an unsupervised model whose goal is to find objects with visual features different than those from the background. Prior efforts have tested simple features such as color and grayscale [4], edges [49] or texture [23], as well as more complex features such as objectness, focusness and backgroundness [68, 148, 22]. The literature offers a wide variety of unsupervised methods working at the pixel level [4], the region level [86], with graph-based methods [51, 145], and with a Bayesian formulation [150]. The reader shall refer to the survey paper by Borji *et al.* [13] for more details on unsupervised methods. While unsupervised methods have their advantages, including simplicity and no need for training, they have been outperformed by machine learning approaches. Although some traditional AI methods such as SVM [136] perform well, deep learning methods, specifically CNN models, have raised the bar and imposed themselves as the unavoidable standard. With CNNs, the saliency problem has been redefined as a labeling problem where feature selection between salient and non-salient objects is done automatically through gradient descent.

CNNs were first developed to perform image classification [80, 8, 83, 81]. These models are made of a series of convolution layers with non-linear activation functions and max pooling operations all the way to a softmax layer which predicts the likelihood of each class. CNN methods are a priori unfit to predict a saliency map since their output is a k-D vector (where k is the number of classes), and not an $N \times M$ map (where $N \times M$ is the size of the input image) as one would expect. However, one can alleviate that problem by extracting a square patch around each pixel and use that patch to predict the center pixel's class [40, 52]. In order for these methods to capture a global context that goes beyond the scope of each patch, they process patches taken from different resolutions of the input image.

Several deep visual saliency detection methods use this same patch trick for predicting a saliency map [94, 157, 87, 143]. Zhao *et al.* [157] integrated the global and local context of an image into a single, multi-context network, where the global context helps to model the saliency in the full image, and the local context helps to estimate the saliency of fine-grained, feature rich areas. Li *et al.* [87] developed a com-

putational model using multi-scale deep features extracted by three CNNs and three fully connected layers to define salient regions of an image. Such a complex model was designed to capture the saliency map of objects with various scales, geometry, spatial positions, irregularities, and contrast levels. Wang *et al.* [143] developed a two tier strategy: each pixel is assigned a saliency based upon a local context estimation in parallel to a global search strategy used to identify the salient regions. These two saliency maps are then combined using geodesic object proposal techniques [78].

Another way of having the output resolution of a CNN match the input image resolution is through one (or several) upsampling layer(s). A popular method for doing so is the FCN method by Long *et al.* [97] which adds an upsampling layer at the very end of the network. Saliency detection methods using that approach are among the most accurate ones [16, 113, 88] most likely because they better capture the local and global context than patch-wise methods.

In order to enforce spatial coherence, a large number of methods use pre-computed regions or super pixels [87, 88, 157, 94, 143]. Roughly speaking, the idea is to set the saliency score of a superpixel as the mean saliency score of each pixel located inside of it. Since superpixels can be inaccurate, some methods [143] use several object proposals which they combine afterwards while others use more than one CNN stream [88, 157]. Spacial coherence can also be enforced by using a CRF or mean-field postprocess [88, 81]. The main inconvenience with these approaches is their processing time.

Our approach differs from these methods as it uses a single and fully convolutional CNN. It uses a series of multiscale convolution and deconvolution blocks organized in a novel 5×4 grid. Our CNN model ensures that the output has the right size while capturing the local and global context as well as features at various resolutions. Spatial coherence is enforced with a loss function inspired by the Mumford-Shah model [111] which we adapted to the context of machine learning.

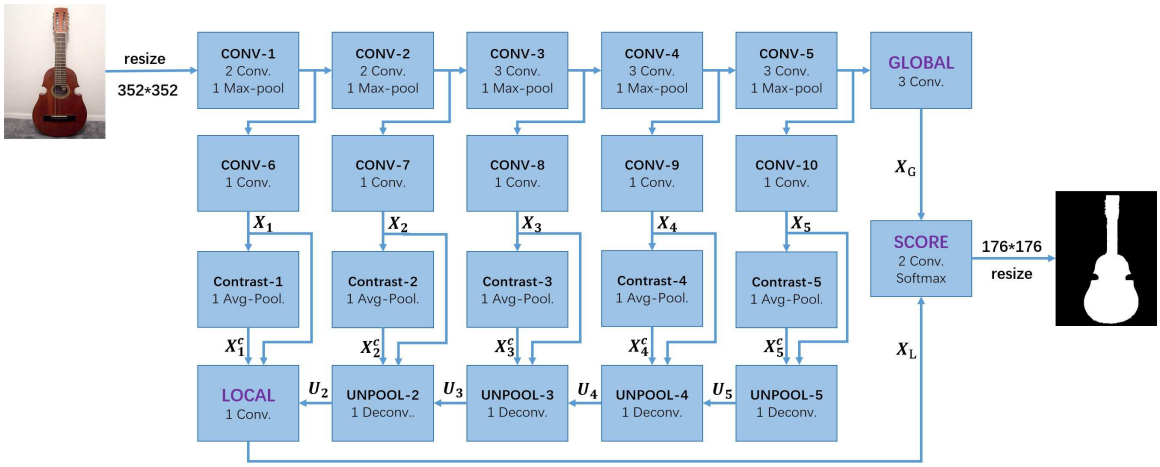

Figure 4.1 – Architecture of our 4×5 grid-CNN network for salient object detection.

4.3 Proposed Method

4.3.1 Model Based Saliency Detection

Salient region detection as well as image segmentation often boils down to the optimization of a non-convex energy function which consists of a data term and a regularization term. An elegant mathematical global model is the cartoon Mumford-Shah (MS) model [111], whose fitting energy,

$$F^{\text{MS}} = \underbrace{\sum_j \lambda_j \int_{\mathbf{v}\in\Omega_j} \left|I(\mathbf{v}) - u_j\right|^2 d\mathbf{v}}_{\text{data fidelity}} + \underbrace{\sum_j \gamma_j \oint_{\mathbf{v}\in C_j} d\mathbf{v}}_{\text{boundary length}} \tag{4.1}$$

segments an image I as a set of disjoint piece-wise constant functions u_j, indexed by j. Here, $\Omega \subset R^N$ is an open set representing the image domain, I is the observed image, u_j is the underlying piece-wise constant segmented image, $\mathbf{v}$ is a pixel location, and C is the boundary of the segmented regions. The positive weighting constants λ_j, and γ_j tune the multi-criteria energy function in terms of data fidelity, and total boundary length. From a Bayesian statistical perspective [15, 161], Eq. (4.1) can be

approximated as,

$$F^{\text{MS}} \approx \underbrace{\sum_j \lambda_j \int_{\mathbf{v}\in\Omega_j} \log p_j(I(\mathbf{v}),\mathbf{v}) d\mathbf{v}}_{\text{data fidelity}} + \underbrace{\sum_j \gamma_j \oint_{\mathbf{v}\in C_j} d\mathbf{v}}_{\text{boundary length}}. \tag{4.2}$$

As there is no analytic solution to Eqs. (4.1) and (4.2), the most common unsupervised approaches to optimize these employ level set base curve evolution techniques [21, 141], generalized Bayesian criteria using the variational principle, and simulated annealing [161]. Despite their mathematical elegance, these methods are all iterative in nature, making them sensitive to initial conditions and likely to fail in the presence of noise, background clutter, weak image boundaries or image non-uniformity. Furthermore, poor convergence rates in the iterative solution of the level set limits their utility to non real-time applications.

To address these issues, we propose a supervised deep convolutional network whose loss approximates the MS functional with the sum of a cross entropy data fidelity term between the ground truth and estimated saliency and a boundary loss term:

$$F^{\text{MS}} \approx \underbrace{\sum_j \lambda_j \int_{\mathbf{v}\in\Omega_j} \text{H}_j(y(\mathbf{v}),\hat{y}(\mathbf{v}))}_{\text{cross entropy}} + \underbrace{\sum_j \gamma_j (1 - \text{IoU}(C_j,\hat{C}_j))}_{\text{boundary IoU loss}} \tag{4.3}$$

where H_j is the total cross entropy between ground truth (y) and estimated ($\hat{y}$) saliency map of all pixels ($\mathbf{v}$) inside region Ω_j, and $\text{IoU}(C_j,\hat{C}_j)$ is the intersection over union between the pixels on the true boundary C_j and the pixels on the estimated boundary $\hat{C}_j$. Note that since our method implements a supervised version of the MS functional, the use of the IoU allows our method to learn a higher level a priori term, i.e. a term that learns to penalize erroneous boundaries instead of minimizing the total boundary length.

4.3.2 Network Architecture

Here we provide a deep convolutional network architecture whose goal is to learn discriminant saliency features (our model is shown in Figure 4.1). As mentioned in

Sec. 4.2, good saliency features must account for both the local and global context of an image and incorporate details from various resolutions. To achieve this goal, we have implemented a novel grid-like CNN network containing 5 columns and 4 rows. Here, each column is geared toward the extraction of features specific to a given input scale. The input I to our model (on the left) is an 352×352 image and the output (on the right) is a 176×176 saliency map which we resize back to 352×352 with a bilinear interpolation.

The first row of our model contains five convolutional blocks derived from VGG-16 [127] (CONV-1 to CONV-5). As shown in Table 4.1, these convolution blocks contain a max pooling operation of stride 2 which down-samples their feature maps $\{X_1, ..., X_5\}$ by a factor of 2, e.g. $\{176 \times 176, 88 \times 88, ..., 11 \times 11\}$. The last and right-most convolution block of the first row computes features X_G that are specific to the global context of the image.

The second and third row is a set of ten convolutional blocks, CONV-6 to CONV-10 for row 2 and Contrast-1 to Contrast-5 for row 3. The aim of these blocks is to compute features (X_i) and contrast features (X_i^C) specific to each resolution. The contrast features capture the difference of each feature against its local neighborhood favoring regions that are either brighter or darker than their neighbors.

The last row is a set of deconvolution layers used to upscale the features maps from 11×11 (bottom right) all the way to 176×176 (bottom left). These UNPOOL layers are a means of combining the feature maps (X_i, X_i^C) computed at each scale. The lower left block constructs the final local feature maps X_L. The SCORE block has 2 convolution layers and a softmax to compute the saliency probability by fusing the local (X_L) and global (X_G) features. Further details of our model are given in Table 4.1.

Non-Local Feature Extraction

Multi-Scale local features: As shown in the second row of Figure 4.1, the convolutional blocks CONV-6 to CONV-10 are connected to the VGG-16 CONV-1 to CONV-5 processing blocks. The goal of these convolutional layers is to learn multi-scale local feature maps $\{X_1, X_2, ..., X_5\}$. Each convolution block has a kernel size 3×3 and 128 channels.

Table 4.1 – Details of the proposed deep convolutional network for predicting salient objects (S: Stride, Pad: zero padding).

Block	Layer	Kernel	S	Pad	Output
CONV-1	2 conv	3*3	1	Yes	352*352*64
	max-pool	2*2	2	Yes	176*176*64
CONV-2	2 conv	3*3	1	Yes	176*176*128
	max-pool	2*2	2	Yes	88*88*128
CONV-3	3 conv	3*3	1	Yes	88*88*256
	max-pool	2*2	2	Yes	44*44*256
CONV-4	3 conv	3*3	1	Yes	44*44*512
	max-pool	2*2	2	Yes	22*22*512
CONV-5	3 conv	3*3	1	Yes	22*22*512
	max-pool	2*2	2	Yes	11*11*512
CONV-6	conv	3*3	1	Yes	176*176*128
CONV-7	conv	3*3	1	Yes	88*88*128
CONV-8	conv	3*3	1	Yes	44*44*128
CONV-9	conv	3*3	1	Yes	22*22*128
CONV-10	conv	3*3	1	Yes	11*11*128
UNPOOL-5	deconv	5*5	2	Yes	22*22*128
UNPOOL-4	deconv	5*5	2	Yes	44*44*256
UNPOOL-3	deconv	5*5	2	Yes	88*88*384
UNPOOL-2	deconv	5*5	2	Yes	176*176*512
LOCAL	conv	1*1	1	No	176*176*640
GLOBAL	conv-1	5*5	1	No	7*7*128
	conv-2	5*5	1	No	3*3*128
	conv-3	3*3	1	No	1*1*128
SCORE	conv-L	1*1	1	No	176*176*2
	conv-G	1*1	1	No	1*1*2

Contrast features: Saliency is the distinctive quality of a foreground object which makes it stand out from its surrounding background. Saliency features must thus be uniform inside the foreground objects and within the background but at the same time be different between foreground and background areas. In order to capture this kind of contrast information, we added a contrast feature associated to each local feature X_i. Each contrast feature X_i^c is computed by subtracting X_i from its local average. The kernel size of the average pooling is 3×3

$$X_i^c = X_i - \text{AvgPool}(X_i). \tag{4.4}$$

Note that such contrast feature is similar in spirit to that of Achanta *et al.* [4] which computes the difference between the pixel RGB color and the global average color of the image. It is even closer to that of Liu and Gleicher [92] which computes contrast features from a Gaussian image pyramid. However, our approach is different as our features are learned and not predefined.

Deconvolution features: Since the size of the final output is 176×176, we use a series of deconvolution layers to increase the size of the precomputed features maps X_i and X_i^C. Instead of increasing the feature maps by a ratio of $\{2, 4, 8, 16\}$ as suggested by Long *et al.* [97] which results in coarse feature maps, we adopt a step-wise upsampling procedure as showed in the third row in Figure 4.1. At each UNPOOL processing block, we upsample the previous feature maps by a factor of 2. The resulting unpooled feature map U_i is computed by combining the information of its local feature X_i, local contrast feature X_i^c, and the previous block's unpooled feature U_{i+1}

$$U_i = \text{UNPOOL}(X_i, X_i^c, U_{i+1}). \tag{4.5}$$

The UNPOOL operation is implemented with a deconvolution layer with a stride of 2 and a 5×5 kernel. The input is the concatenation of X_i, X_i^C and U_{i+1} . The number of feature channels of U_i is equal to the sum of X_i and U_{i+1}.

Local feature maps: We use a convolution layer with a kernel size 1×1 to get the final local feature maps X_L. The input of that layer is the concatenation of X_1, X_1^C

and U_2

$$X_L = \text{CONV}(X_1, X_1^c, U_2). \tag{4.6}$$

The number of feature channels of X_L is equal to the sum of X_1 and U_2. Note that we tried using another UNPOOL operation to increase the size of X_L from 176×176 to 352×352, but found that this operation doubles the computation time without measurably improving accuracy.

Capturing global context: Detecting salient objects in an image requires the model to capture the global context of the image before assigning saliency to individual small regions. To account for this, we added three convolutional layers after the CONV-5 block to compute the global feature X_G. The first two convolutional layers have a kernel size of 5, and the last convolutional is 3. All three layers have 128 features channels.

4.3.3 Cross Entropy Loss

The final saliency map is computed as a linear combination of the local features X_L and global features X_G using two linear operators $(W_L,\, b_L)$ and $(W_G,\, b_G)$. The softmax function is used to compute the probability for each pixel of being salient or not.

$$\hat{y}(\mathbf{v}) = p\Big(y(\mathbf{v}) = c\Big) = \frac{e^{W_L^c X_L(\mathbf{v}) + b_L^c + W_G^c X_G + b_G^c}}{\sum_{c' \in \{0,1\}} e^{W_L^{c'} X_L(\mathbf{v}) + b_L^{c'} + W_G^{c'} X_G + b_G^{c'}}} \tag{4.7}$$

The cross-entropy loss function

$$H_j(y(\mathbf{v}), \hat{y}(\mathbf{v})) = -\frac{1}{N} \sum_{i=1}^{N} \sum_{c \in \{0,1\}} (y(\mathbf{v}_i) = c) \Big(\log\Big(\hat{y}(\mathbf{v}_i) = c\Big)\Big) \tag{4.8}$$

is used to minimize the first data term in Eq. (4.2).

4.3.4 IoU Boundary Loss

Motivated by the significant applications of Dice loss or IoU boundary loss in medical image segmentation [164, 133, 109], our proposed method approximates the penalty on boundary length of Eq. (4.1) using an IoU boundary loss term. To compute

the boundary loss, we approximate the saliency map gradient magnitude (and hence the boundary pixels) using a Sobel operator followed by a tanh activation. The tanh activation projects the gradient magnitude of saliency maps to a probability range of $[0, 1]$. Given the gradient magnitude of saliency maps $\hat{C}_j$ and gradient magnitude of true saliency maps C of region j, the Dice or IoU boundary loss can be computed as

$$\text{IoU Loss} = 1 - \frac{2|C_j \cap \hat{C}_j|}{|C_j| + |\hat{C}_j|}, \tag{4.9}$$

which has range [0, 1]. Our whole boundary overlapping loss computation procedure is end-to-end trainable, and an example is shown in Figure 4.2. Please note that the intersection is implemented using a point-wise multiplication operator.

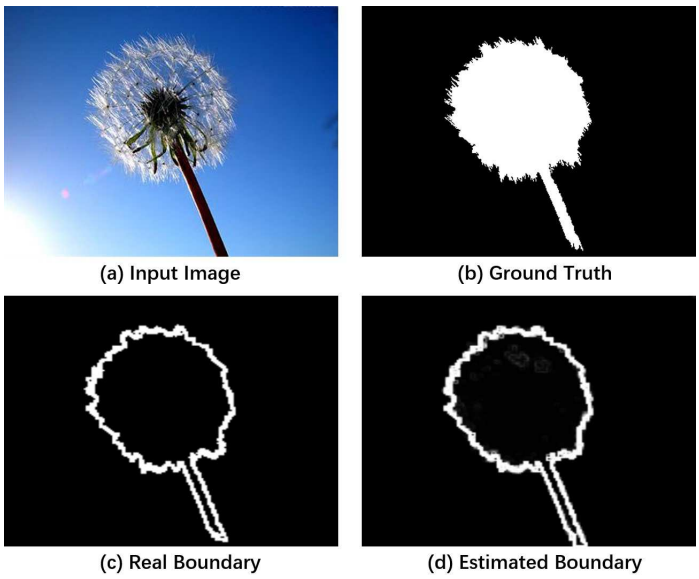

Figure 4.2 – A single input image (a) together with its groundtruth saliency (b) and boundary (c) is used to train a model only containing the IoU boundary loss term in Eq. (4.3). The estimated boundary (d) after training for 200 iterations is in excellent agreement with the true boundary.

4.4 Experimental Results

4.4.1 Benchmark Datasets

We have evaluated the performance of our method (NLDF) on six different public benchmark datasets: MSRA-B [95], HKU-IS [87], DUT-OMRON [152], PASCAL-S [90], ECSSD [151] and SOD [105].

MSRA-B: contains 5000 images, and is widely used for visual saliency detection. Most of the images have one salient object and a pixel-wise ground truth [67].

HKU-IS: contains 4447 images, most of which have low contrast and multiple salient objects. This dataset has been split into 2500 training images, 500 validation images and the remaining 1447 test images.

DUT-OMRON: contains 5168 challenging images, each of which contains one or more salient objects with a relatively cluttered background.

PASCAL-S: contains 850 natural images which were built from the validation set of the PASCAL-VOC 2010 segmentation challenge. This dataset contains both pixel-wise saliency ground truth and eye fixation ground truth labeled by 12 subjects.

ECSSD: contains 1000 images with complex structure acquired from the Internet. The ground truth masks were labeled by 5 subjects.

SOD: contains 300 images originally designed for image segmentation. Many images contain multiple salient objects with low contrast and overlapping boundaries.

4.4.2 Implementation and Experimental Setup

Our NLDF model was implemented in TensorFlow [1]. The weights in the CONV-1 to CONV-5 blocks were initialized with the pretrained weights of VGG-16 [127]. All the weights of newly added convolution and deconvolution layers were initialized randomly with a truncated normal ($\sigma = 0.01$), and the biases were initialized to 0. The Adam optimizer [76] was used to train our model with an initial learning rate of 10^{-6}, $\beta_1 = 0.9$, and $\beta_2 = 0.999$. The λ_j and γ_j in Eq. (4.3) were set to 1.

For fair comparison with other methods, we followed the experimental setup of [67], dividing the MSRA-B dataset into 3 parts: 2500 images for training, 500 images for validation and the remaining 2000 images for testing. The training and validation

Table 4.2 – Quantitative performance of our model on six benchmark datasets compared with the GS [148], MR [152], wCtr* [162], BSCA [117], LEGS [143], MC [157], MDF [87] and DCL [88]models. The latter four are deep learning methods and the former are not. The F_β and MAE metrics are defined in the text.

Dataset	Metric	GS	MR	wCtr*	BSCA	LEGS	MC	MDF	DCL	DCL+	**NLDF-**	**NLDF**
MSRA-B	max F_β	0.777	0.824	0.820	0.830	0.870	0.894	0.885	0.905	**0.916**	0.912	0.911
	MAE	0.144	0.127	0.110	0.130	0.081	0.054	0.066	0.052	**0.047**	0.048	0.048
HKU-IS	max F_β	0.682	0.715	0.726	0.723	0.770	0.798	0.861	0.892	**0.904**	0.874	0.902
	MAE	0.167	0.174	0.141	0.174	0.118	0.102	0.076	0.054	0.049	0.060	**0.048**
DUT-OMRON	max F_β	0.557	0.610	0.630	0.616	0.669	0.703	0.694	0.733	**0.757**	0.724	0.753
	MAE	0.173	0.187	0.144	0.191	0.133	0.088	0.092	0.084	**0.080**	0.085	**0.080**
PASCAL-S	max F_β	0.624	0.666	0.659	0.666	0.756	0.740	0.764	0.815	0.822	0.804	**0.831**
	MAE	0.224	0.223	0.201	0.224	0.157	0.145	0.145	0.113	0.108	0.116	**0.099**
ECSSD	max F_β	0.661	0.736	0.716	0.758	0.827	0.822	0.832	0.887	0.901	0.886	**0.905**
	MAE	0.206	0.189	0.171	0.183	0.118	0.106	0.105	0.072	0.075	0.075	**0.063**
SOD	max F_β	0.601	0.619	0.632	0.634	0.707	0.688	0.745	0.795	0.801	0.776	**0.810**
	MAE	0.266	0.273	0.245	0.266	0.215	0.197	0.192	**0.142**	0.153	0.161	0.143

sets were combined together to train our model with horizontal flipping as data augmentation. The inputs were resized to 352×352 for training. With an NVIDIA Titan X GPU, it takes ~ 9 hours to finish the whole training procedure for 20 epochs with a single image batch size. Without further optimization, this trained model was used to compute the saliency maps of the other datasets.

4.4.3 Evaluation Criteria

Precision-recall (PR) curves, F_β and mean absolute error (MAE) were used as metrics to evaluate the performance of saliency detection. The PR curve is computed by binarizing the saliency maps under different probability thresholds ranging from 0 to 1 and comparing against the ground truth. As for the F_β measure, it is defined as,

$$F_\beta = \frac{(1+\beta^2) \cdot \text{Precision} \cdot \text{Recall}}{\beta^2 \cdot \text{Precision} + \text{Recall}}. \tag{4.10}$$

where $\beta^2 = 0.3$ to emphasize precision over recall as suggested in [4]. We report the maximum F-Measure computed from the PR curve. MAE [115] is computed as the average pixel-wise absolute difference between the estimated saliency map S and its corresponding ground truth L,

$$\text{MAE} = \frac{1}{W \times H} \sum_{x=1}^{W} \sum_{y=1}^{H} \Big| S(x,y) - L(x,y) \Big|. \tag{4.11}$$

where W and H is the width and height of a given image.

4.4.4 Effectiveness of the Boundary Loss Term

In addition to our NLDF model, we also trained a model, denoted as NLDF-, which only contains the cross-entropy loss term and excludes the boundary loss term [see Eq. 4.3]. As shown in Figure 4.3, the saliency maps generated from NLDF- are fairly coarse and the boundary of the salient objects are not well preserved. As shown in last two columns of Table 4.2, this qualitative decrease in performance is also mirrored in the quantitative results. The inclusion of the boundary loss in NLDF as compared to NLDF- accounts for increases in max F_β of 2.1% to 4.4% and decreases in MAE of

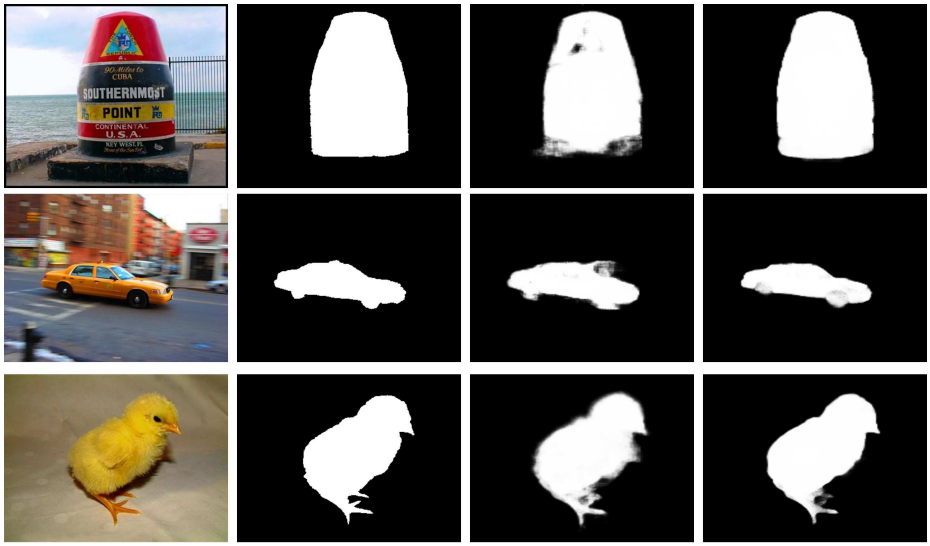

Figure 4.3 – Visual comparison of saliency detection results with and without the boundary loss term in Eq. (4.2).

5.8% to 20.0% on HKU-IS, DUT-OMRON, PASCAL-S, ECSSD and SOD datasets. Little change is observed for MSRA-B, an expected result, since training and testing samples are drawn from a similar pool of images. Significantly, these results illustrate that the boundary loss term directly enhances the generality of NLDF, making it more robust to variations in input types.

4.4.5 Comparison with the State of the Art

We quantitatively compared our NLDF method with several recent state-of-the-art methods: Geodesic Saliency (GS) [148], Manifold Ranking (MR) [152], optimized Weighted Contrast (wCtr*) [162], Background based Single-layer Cellular Automata (BSCA) [117], Local Estimation and Global Search (LEGS) [143], Multi-Context (MC) [157], Multiscale Deep Features (MDF) [87] and Deep Contrast Learning (DCL) [88]. LEGS, MC, MDF and DCL are the latest deep learning based saliency detection methods. Note that since part of the HKU-IS dataset was used to train the MDF model [87], we only compute the evaluation metrics on the testing set of HKU-IS. Also the MDF only provided 200 pre-compute saliency maps on SOD dataset, we

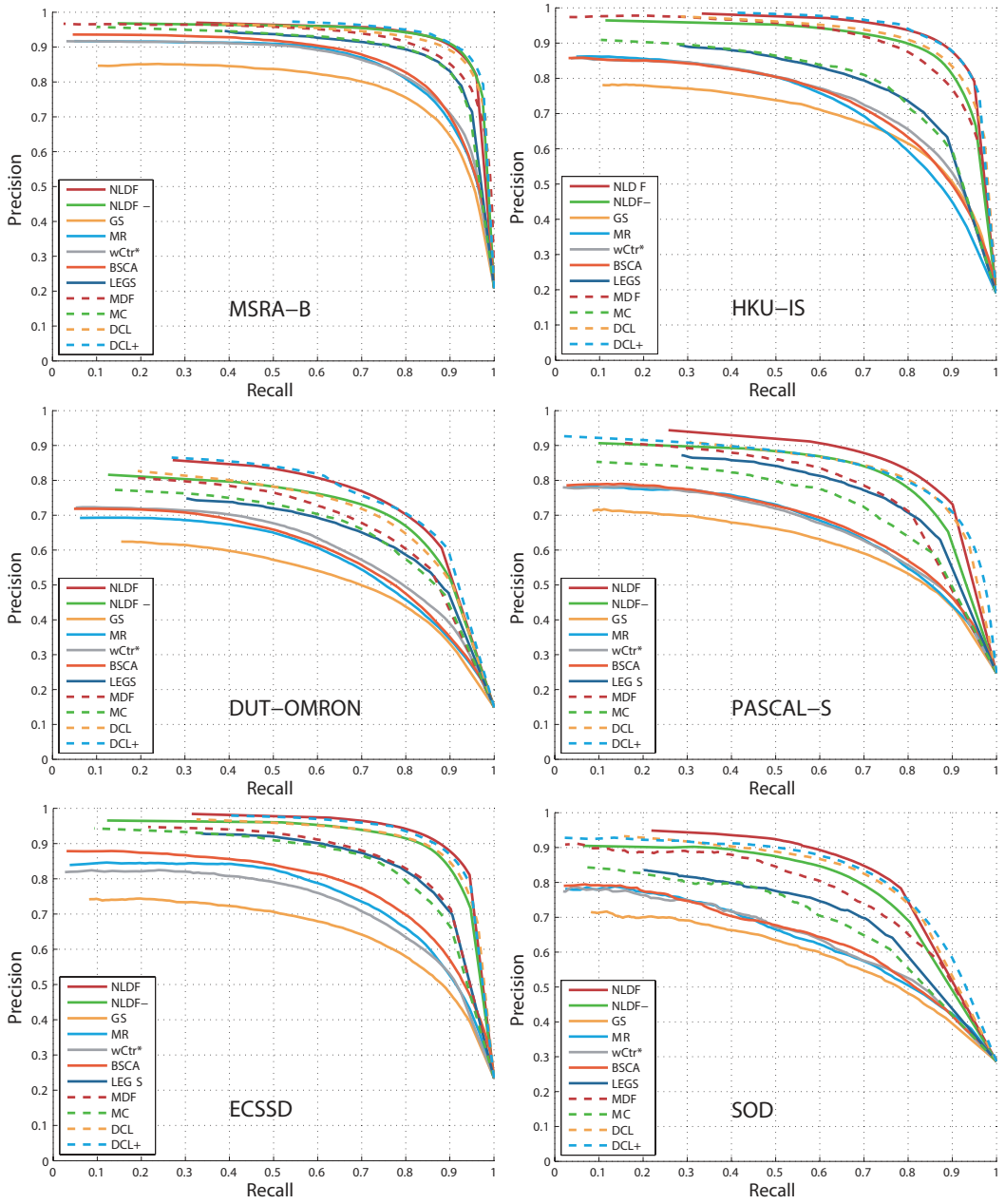

Figure 4.4 – Precision-recall curves for our model compared to GS [148], MR [152], wCtr* [162], LEGS [143], BSCA [117], MDF [87], MC [157] and DCL [88] evaluated on the MASR-B, HKU-IS, DUT-OMRON, PASCAL-S, ECSSD and SOD benchmark datasets. Our NLDF model can deliver state-of-the-art performance on all six datasets.

use the same subset for evaluation.

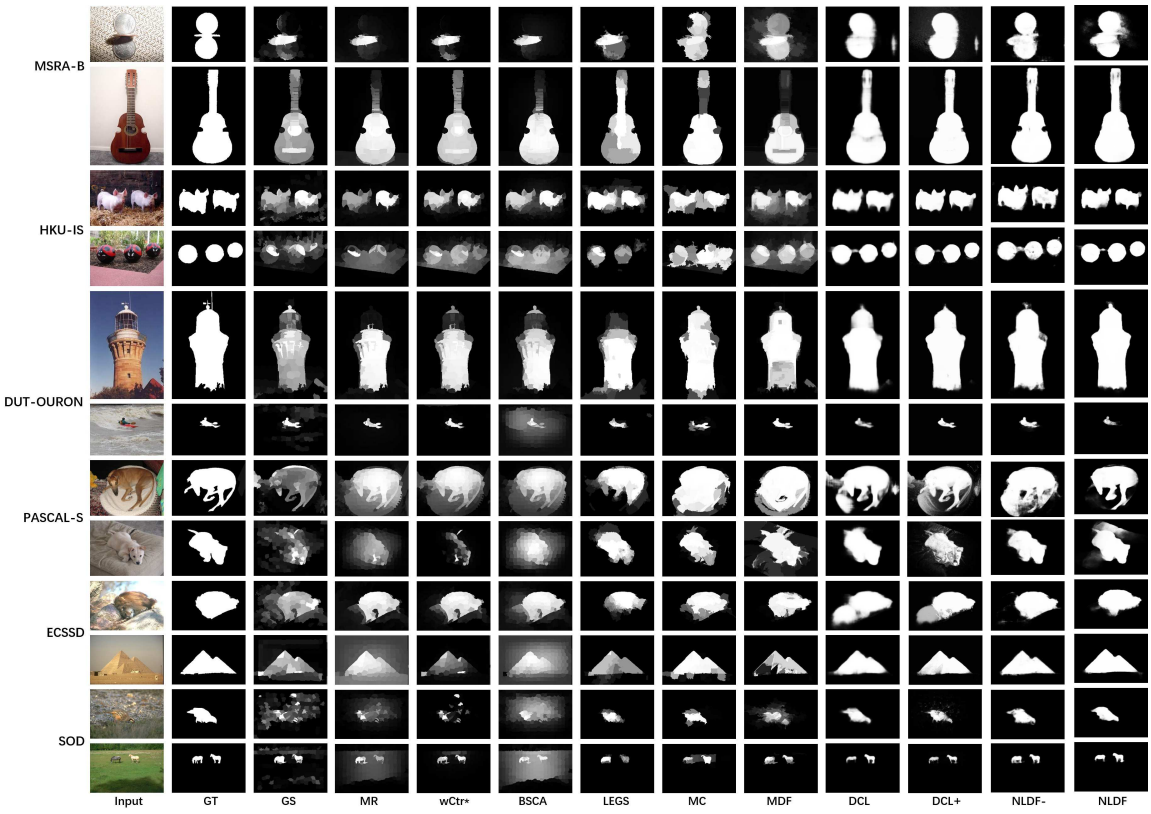

Figure 4.5 – Saliency maps produced by the GS [148], MR [152], wCtr* [162], BSCA [117], LEGS [143], MC [157], MDF [87] and DCL [88] methods compared to our NLDF method. The NLDF maps provides clear salient regions and exhibit good uniformity as compared to the saliency maps from the other deep learning methods (LEGS, MC, MDF and DCL). Our method is also more robust to background clutter than the non-deep-learning methods (GS, MR, wCtr* and BSCA).

In comparison to the top performing method, DCL+, [88] an extension of DCL that uses a fully-connected CRF [77] as a post-processing step to refine the saliency map, we find that NLDF attains nearly identical (or better) performance across the board (see Table 4.2). That this is achieved without a significant post-processing step means that the execution time and implementation complexity are greatly reduced. The computation time reported in [88] for DCL is 1.5 s per (300×400) image and an additional 0.8 s for CRF post-processing (DCL+). In comparison, our NLDF

method only requires 0.08 s per image on a Titan X GPU. This substantial speedup enables nearly real-time salient object detection while also delivering state-of-the-art performance.

A visual comparison of the saliency maps is provided in Figure 4.5. All saliency maps of other methods were either provided by the authors or computed using the authors' released code. Precision-recall curves are shown in Figure 4.5 and the maximum F_β and MAE scores are in Table 4.2. As shown in Table 4.2, our NLDF model achieves superior quantitative max F_β, MAE and PR performance across the board when compared to GS, MR, wCtr*, BSCA, LEGS, MC, MDF and DCL. NLDF also surpasses DCL+ more times than not in max F_β and MAE and exhibits equivalent or better PR curves.

We also compared the average computation time with other four leading deep learning methods for generating the saliency map of one images in Table 4.3. On a Titan Black GPU, our approach is 18 to 100 times faster than existing methods.

Table 4.3 – Inference time of leading deep learning methods.

	LEGS	MC	MDF	DCL	DCL+	**NLDF**
s/img	2	1.6	8	1.5	2.3	**0.08**

4.5 Experiments on Traffic Analysis

4.5.1 MIO-TCD dataset

As our proposed model is a general purpose image segmentation model and vehicles can be considered as salient objects in traffic scenes, we also utilize this model to do traffic analysis (foreground vehicle segmentation). The MIO-TCD dataset was used for training (110,000 images) and testing (27,743 images). The beauty of this dataset is that each vehicle has been labeled with an outline boundary and a category label, which enable us to generate pixel-wise accurate foreground maps.

We trained our NLDF model without the boundary term on the training set of MIO-TCD dataset for 10 epochs, and the whole training process needs around two days. After training, the testing set was used for testing. In Figure 4.6, we plot the

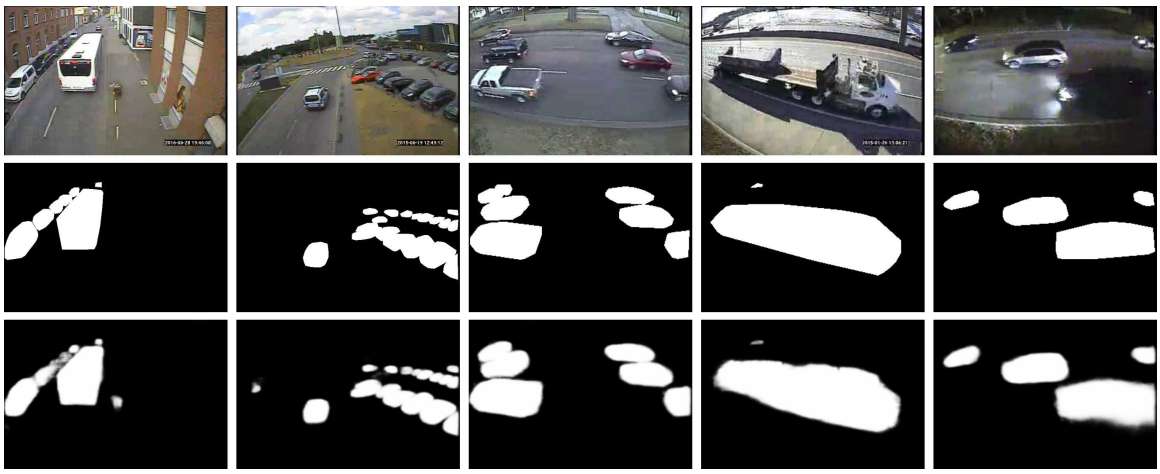

Figure 4.6 – The foreground vehicle segmentation result of using our model on the MIO-TCD dataset. The first row are the input images, the second row are the ground truth, and the third are the results produced by our model.

segmentation results of our model. As can be seen, our model can precisely segment vehicles. Most importantly, our model barely produces false segmentation which is very useful for our next vehicle localization and classification task.

For quantitative evaluation, we compute the accuracy of the segmentation results (that is the number of correctly labeled pixels divided by the total number of pixels), as well as the two metrics ($\max F_\beta$ and MAE) used for the evaluation of saliency detection.

Table 4.4 – The Accuracy on MIO-TCD dataset of our model

	Accuracy	$\max F_\beta$	MAE
Ours	99.3%	0.9522	0.0088

4.5.2 CDnet 2014 Dataset

In order to test the generalization ability of the model trained on the MIOTCD dataset for processing other traffic videos, we select 9 traffic video sequences from the CDnet 2014 dataset for testing. These videos are with various challenge conditions such as low frame-rate, night video, camera jitters and pan tilt zoom. Some of sample

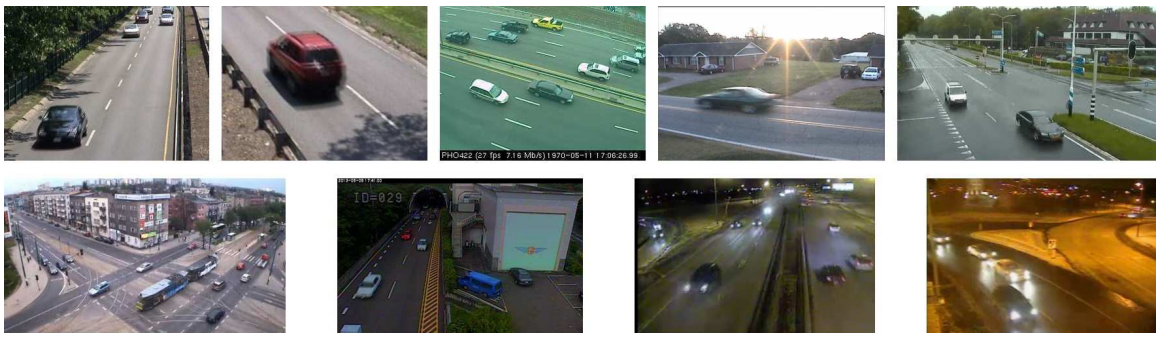

Figure 4.7 – Sample video frames of the 9 traffic videos from the CDnet 2014 dataset. Videos in the first row are 'highway', 'traffic', 'turnpike', 'continuousPan' and 'twoPositionPTZCam'. The second row are 'tramCrossroad', 'tunnelExit', 'fluidHighway' and 'winterStreet'.

frames are showed in Figure 4.7. We process each video one frame by one frame, and then threshold the foreground probability map by using a threshold 0.9.

Four top-ranking methods from the changedetection.net were selected for comparison with our model, there are SubSENSE [131], PAWCS [130], FTSG [146] and IUTIS-5 [10]. The following 7 metrics provided by the CDnet 2014 dataset is used for evaluation: Recall, Specificity, False Positive Rate(FPR), False Negative Rate(FNR), Percentage of Wrong Classifications(PWC), Precision and F-Measure. The quantitative comparison is given in Table 4.5. As can be seen, the model trained on MIO-TCD dataset can reach the top performance for analyzing traffic videos from the CDnet 2014 dataset. This means our proposed model trained on MIO-TCD dataset can generalize well for analyzing various kinds of traffic scenes.

A visual comparison of the foreground detection results are provided in Figure 4.8. Overall our model can detected almost all the moving vehicles, but may miss some of small cars as showed in the "tramCrossRoad" video. Due to our model is trained on the MIO-TCD dataset, another big issue is the false positive as showed in the "traffic" and "tunnelExit" video, and also our model detected the non-moving vehicles such as the right part in the "continuousPan" video. As the MIO-TCD contains video frames of the night, our methods can perfectly localize those moving vehicles for two night video sequences "fluidHighway" and "winterStreet", while all the other background

subtraction methods cannot correctly find the moving vehicles due to bad illumination conditions.

Table 4.5 – The evaluation metrics computed on the traffic videos from the CDnet 2014 dataset.

Method	**Recall**	**Specificity**	**FPR**	**FNR**	**PWC**	**Precision**	**F-Measure**
Ours	0.787	0.986	0.014	0.213	2.072	0.679	0.690
SubSENSE	0.815	0.982	0.018	0.185	2.416	0.633	0.681
PAWCS	0.730	0.992	0.008	0.270	1.781	0.713	0.689
FTSG	0.749	0.982	0.018	0.251	2.497	0.648	0.662
IUTIS-5	0.788	0.987	0.013	0.212	1.855	0.682	0.718

4.6 Conclusion

The integration of local and global features has already been shown to be a powerful mechanism for saliency detection. Here we took this approach one step further by adding a boundary loss term to the typical cross entropy loss, in effect implementing the Mumford-Shah functional in a deep neural net framework and training it end-to-end. The resulting model achieves state-of-the-art performance across multiple saliency detection benchmark datasets, does not use any special pre- or post-processing steps and computes saliency maps 18 to 100 times faster than competing systems.

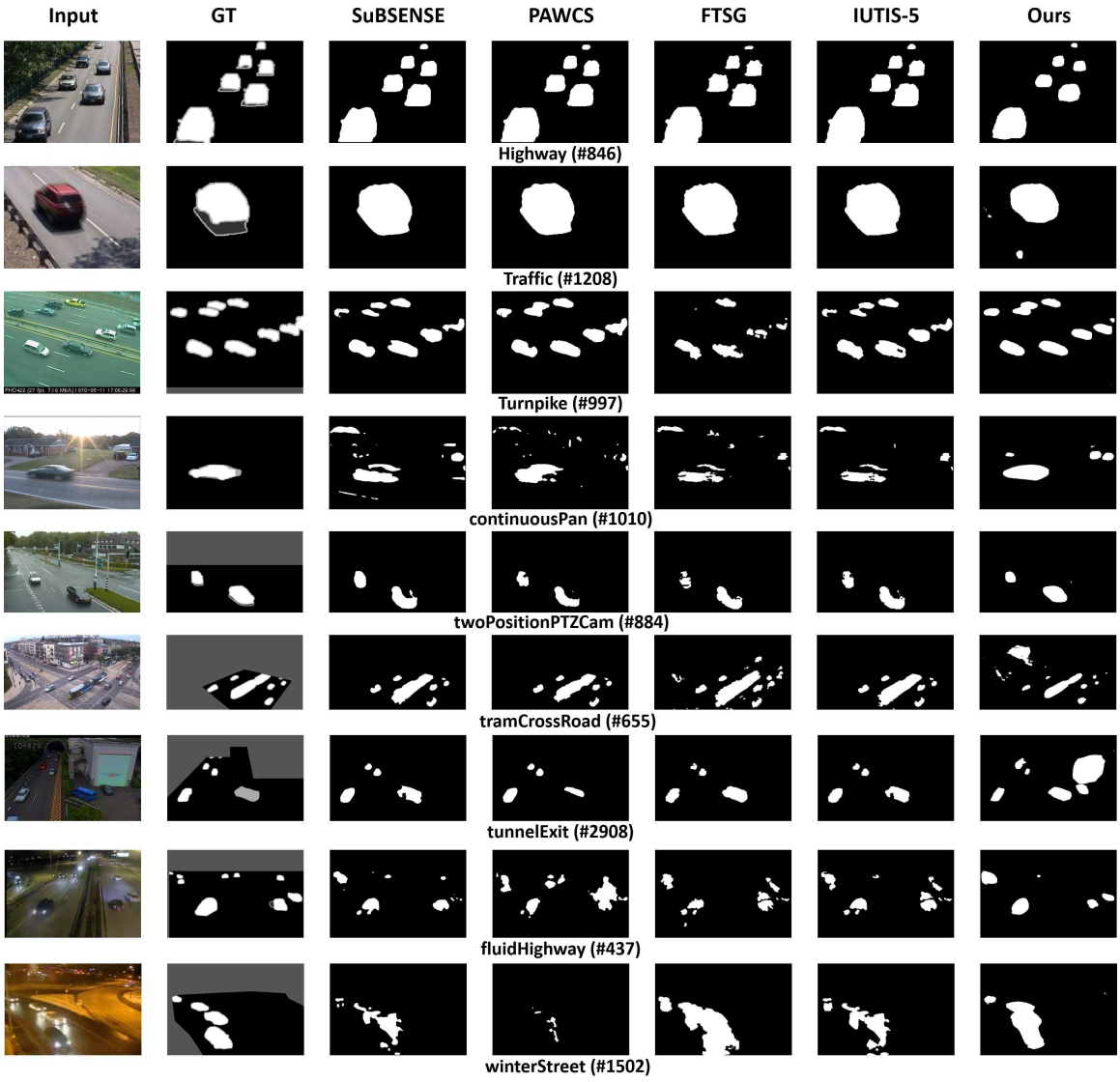

Figure 4.8 – The foreground detection maps produced by the SubSENSE [131], PAWCS [130], FTSG [146], IUTIS-5 [10] and our model trained on MIO-TCD dataset.

Chapter 5

Conclusion

5.1 Summary

In this thesis, we focused on two aspects of traffic analysis without using motion features with low frame-rate videos: Traffic density flow analysis [100] and Vehicle detection and classification [101, 103].

Traffic density flow analysis: we investigated the possibility of monitoring highway traffic based on videos whose frame rate is too low to accurately estimate motion features. We proposed several CNN models to segment traffic images into three different classes (road, car and background), classify traffic images into different categories (empty, fluid, heavy, jam) and predict traffic density without using motion features. In order to generalize the model trained on a specific dataset to analyze new traffic scenes, we also proposed a novel transfer learning framework to do model adaptation.

Vehicle detection and classification: in collaboration with colleagues from Miovision inc. (Waterloo, On), we built and released the largest traffic dataset in the world. This dataset (MIO-TCD) is dedicated to vehicle localization and classification. Based on this dataset, we organized the Traffic Surveillance Workshop and Challenge in conjunction with CVPR 2017 and built an online evaluation system.

Secondly, we evaluated several state-of-the-art deep learning methods for the classification and localization task on the MIO-TCD dataset. In light of the results, we may conclude that state-of-the-art deep learning methods exhibit a capacity to localize and recognize vehicle from single video frames. Through a thorough analysis of the failure cases of those methods, we find that methods are very likely to get confused between categories with similar visual characteristics on both the classification and localization datasets, also rare classes are more likely to be miss-classified, and small objects suffer from localization error due to small scales.

Lastly, as saliency detection aims to highlight the most relevant objects in an image (e.g. vehicles in traffic scenes), we proposed a multi-resolution 4×5 grid CNN model for the salient object detection. The model enables near real-time high performance saliency detection. We also extend this model to do traffic analysis, experiments demonstrate our model can do precisely foreground vehicle segmentation.

5.2 Future works

Despite recent rapid progress of deep learning methods applied to traffic analysis, these are still many areas I would like to explore after my Ph.D study:

Instance-level segmentation: In Chapter 4, we proposed a CNN model for salient object segmentation and the model can achieve excellent performances for foreground vehicle segmentation. The limits of our model is it only can detection salient regions, and, as such, is unable to identify different instances. I would like to extend this model to do an instance-level segmentation and also classify different instances into different sub-categories.

Model compression: As we know, a deep model usually contains millions of parameters and is both computationally and memory intensive. I would like to design algorithms to compress deep models, making them deployable on an embedded mobile hardware. Then the traffic surveillance cameras only need to send useful information to the server for making a deep analysis which can save the transmission bandwidth.

Large scale traffic surveillance network analysis: Right now, our works were only focused on analyzing traffic data for individual traffic cameras. While roads are all connected together, analyzing data from a large scale traffic surveillance network could play a significant role in intelligent traffic surveillance systems. There are lots of potential research spots, such as:

— In Chapter 2, we showed that CNN can be used to estimate the traffic density in a single images. From this previous work, we could extend the model to analyze the time series data obtained from a camera network, and by there get a better understanding of how traffic density is evolving in time. Then we can provide accurate and timely traffic congestion information for each individual traveler, the business sectors and government agencies.

— As shown in Chapter 3, deep learning based object detection methods can precisely located moving vehicles in various traffic cameras, to go a further step, it's possible to train another deep model to re-identify the same vehicle across different cameras, which can be used to discover, locate, and track a target vehicle.

Appendix A

Publications

1. **Luo Z.**, B.-Charron F., Lemaire C., Konrad J., Li S., Mishra A., Achkar A., Eichel J., Jodoin P.-M., "*MIO-TCD: A new benchmark dataset for vehicle classification and localization*", submitted to IEEE Transactions on Image Processing, 2017
2. **Luo Z.**, Mishra A., Achkar A., Eichel J., Li S.-Z., Jodoin P.-M., "*Non-Local Deep Features for Salient Object Detection*", IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2017).
3. **Luo Z.**, Jodoin P.-M., Su S.-Z., Li S.-Z., Larochelle H., "*Traffic Analytics with Low Frame Rate Videos*", in Press at IEEE Transactions on Circuits and Systems for Video Technology, 2016.
4. **Luo Z.**, Jodoin P.-M., Li S.-Z., Su S.-Z., "*Traffic Analysis without Motion Features*", IEEE International Conference on Image Processing (ICIP 2015).
5. Zotti C., **Luo Z.**, Lalande A., Humbert O., Jodoin P.-M., "*GridNet with automatic shape prior registration for automatic MRI cardiac segmentation*", MICCAI - ACDC Challenge, 2017
6. Wang Y, **Luo Z**, Jodoin P.-M., "**Interactive Deep Learning Method for Segmenting Moving Objects**", Pattern Recognition Letter, 2016.

Bibliography

[1] M. Abadi and et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL http://tensorflow.org/. Software available from tensorflow.org.

[2] N. Abbas, M. Tayyab, and M. T. Qadri. Real time traffic density count using image processing. *Int. J. Comput. Appl.*, 83(9), 2013.

[3] R. Achanta and S. Süsstrunk. Saliency detection for content-aware image resizing. In *Proc. ICIP*, pages 1005–1008, 2009.

[4] R. Achanta, S. Hemami, F. Estrada, and S. Susstrunk. Frequency-tuned salient region detection. In *Proc. CVPR*, pages 1597–1604, 2009.

[5] J. M. Alvarez, T. Gevers, Y. LeCun, and A. M. Lopez. Road scene segmentation from a single image. In *Proc. ECCV*, pages 376–389. 2012.

[6] O. Asmaa, K. Mokhtar, and O. Abdelaziz. Road traffic density estimation using microscopic and macroscopic parameters. *Image and Vision Computing*, 31(11): 887–894, 2013.

[7] E. Bas, A. Tekalp, and F. Salman. Automatic vehicle counting from video for traffic flow analysis. In *Proc. IEEE Intell. Vehic. Symp.*, pages 392–397, 2007.

[8] Y. Bengio, A. Courville, and P. Vincent. Representation learning: A review and new perspectives. *IEEE Trans. Pattern Anal. Mach. Intell.*, 35(8):1798–1828, 2013.

[9] D. Beymer, P. McLauchlan, B. Coifman, and J. Malik. A real-time computer vision system for measuring traffic parameters. In *Proc. CVPR*, pages 495–501, 1997.

[10] S. Bianco, G. Ciocca, and R. Schettini. How far can you get by combining change detection algorithms? *arXiv preprint arXiv:1505.02921*, 2015.

[11] C. M. Bishop. *Pattern recognition and machine learning*. springer, 2006.

[12] BITRE. New traffic data sources – an overview. *Bureau of Infrastructure, Transport and Regional Economics (BITRE), Australia*, 2014.

[13] A. Borji, M.-M. Cheng, H. Jiang, and J. Li. Salient object detection: A survey. *arXiv preprint arXiv:1411.5878*, 2014.

[14] J. S. Bridle. Training stochastic model recognition algorithms as networks can lead to maximum mutual information estimation of parameters. In *Advances in neural information processing systems*, pages 211–217, 1990.

[15] T. Brox and D. Cremers. On local region models and a statistical interpretation of the piecewise smooth mumford-shah functional. *Int. J. Comput. Vision*, 84 (2):184–193, Jun. 2009.

[16] N. Bruce, C. Catton, and S. Janjic. A deeper look at saliency: Feature contrast, semantics, and beyond. In *Proc. CVPR*, pages 516–524, 2016.

[17] C.-A. Brust, S. Sickert, M. Simon, E. Rodner, and J. Denzler. Convolutional patch networks with spatial prior for road detection and urban scene understanding. *arXiv preprint arXiv:1502.06344*, 2015.

[18] N. Buch, S. A. Velastin, and J. Orwell. A review of computer vision techniques for the analysis of urban traffic. *IEEE Trans. Intell. Transp. Syst.*, 12(3):920–939, 2011.

[19] Z. Cai, Q. Fan, R. S. Feris, and N. Vasconcelos. A unified multi-scale deep convolutional neural network for fast object detection. In *Proc. ECCV*, pages 354–370, 2016.

[20] A. Chan and N. Vasconcelos. Classification and retrieval of traffic video using auto-regressive stochastic processes. In *Proc. IEEE Intell. Vehic. Symp.*, pages 771–776, 2005.

[21] T. F. Chan and L. A. Vese. Active contours without edges. *IEEE Trans. Image Process.*, 10(2):266–277, 2001.

[22] K.-Y. Chang, T.-L. Liu, H.-T. Chen, and S.-H. Lai. Fusing generic objectness and visual saliency for salient object detection. In *Proc. ICCV*, pages 914–921, 2011.

[23] Z. Chen, Y. Liu, B. Sheng, J.-N. Liang, J. Zhang, and Y.-B. Yuan. Image saliency detection using gabor texture cues. *Multimedia Tools and Applications*, pages 1–16, 2015.

[24] F. Chollet et al. Keras. https://github.com/fchollet/keras, 2015.

[25] F. Chollet. Xception: Deep learning with depthwise separable convolutions. *arXiv preprint arXiv:1610.02357*, 2016.

[26] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proc.CVPR*, pages 3213–3223, 2016.

[27] C. Cortes and V. Vapnik. Support-vector networks. *Machine learning*, 20(3): 273–297, 1995.

[28] R. Cucchiara, C. Grana, M. Piccardi, and A. Prati. Detecting moving objects, ghosts, and shadows in video streams. *IEEE Trans. Pattern Anal. Mach. Intell.*, 25(10):1337–1342, 2003.

[29] J. Dai, Y. Li, K. He, and J. Sun. R-FCN: Object detection via region-based fully convolutional networks. *arXiv preprint arXiv:1605.06409*, 2016.

[30] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *Proc. CVPR*, pages 886–893, 2005.

[31] T. Darwish and K. A. Bakar. Traffic density estimation in vehicular ad hoc networks: A review. *Ad Hoc Networks*, 24:337–351, 2015.

[32] K. Derpanis and R. Wildes. Classification of traffic video based on a spatiotemporal orientation analysis. In *IEEE Workshop on Applications of Computer Vision (WACV)*, pages 606–613, 2011.

[33] P. Dollar, C. Wojek, B. Schiele, and P. Perona. Pedestrian detection: An evaluation of the state of the art. *IEEE Trans. Pattern Anal. Mach. Intell.*, 34(4): 743–761, 2012.

[34] Z. Dong, Y. Wu, M. Pei, and Y. Jia. Vehicle type classification using a semisupervised convolutional neural network. *IEEE Trans. Intell. Transp. Syst.*, 16 (4):2247–2256, 2015.

[35] Z. Dong, Y. Wu, M. Pei, and Y. Jia. Vehicle type classification using a semisupervised convolutional neural network. *IEEE Trans. Intell. Transp. Syst.*, 16 (4):2247–2256, 2015.

[36] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *J. Mach. Learn. Res.*, 12:2121–2159, 2011.

[37] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman. The pascal visual object classes (voc) challenge. *Int. J. Comput. Vision*, 88(2): 303–338, 2010.

[38] M. Everingham, S. A. Eslami, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman. The pascal visual object classes challenge: A retrospective. *Int. J. Comput. Vision*, 111(1):98–136, 2015.

[39] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. Liblinear: A library for large linear classification. *J. Mach. Learn. Res.*, 9:1871–1874, 2008.

[40] C. Farabet, C. Couprie, L. Najman, and Y. LeCun. Learning hierarchical features for scene labeling. *IEEE Trans. Pattern Anal. Mach. Intell.*, 35(8):1915–1929, 2013.

[41] K. Garg, S.-K. Lam, T. Srikanthan, and V. Agarwal. Real-time road traffic density estimation using block variance. In *IEEE Winter Conf. Applic. of Comput. Vis.*, pages 1–9, 2016.

[42] A. Geiger, P. Lenz, and R. Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Proc. CVPR*, pages 3354–3361, 2012.

[43] R. Girshick. Fast R-CNN. In *Proc. ICCV*, pages 1440–1448, 2015.

[44] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proc. CVPR*, pages 580–587, 2014.

[45] W. N. Gonçalves, B. B. Machado, and O. M. Bruno. A complex network approach for dynamic texture recognition. *Neurocomputing*, 153:211–220, 2015.

[46] N. Goyette, P. Jodoin, F. Porikli, J. Konrad, and P. Ishwar. Changedetection.net: A new change detection benchmark dataset. In *Proc. CVPRW*, pages 1–8, 2012.

[47] R. Guerrero-Gómez-Olmedo, R. J. López-Sastre, S. Maldonado-Bascón, and A. Fernández-Caballero. Vehicle tracking by simultaneous detection and viewpoint estimation. In *International Work-Conference on the Interplay Between Natural and Artificial Computation*, pages 306–316, 2013.

[48] R. Guerrero-Gómez-Olmedo, B. Torre-Jiménez, R. López-Sastre, S. Maldonado-Bascón, and D. Oñoro-Rubio. Extremely overlapping vehicle counting. In *Iberian Conference on Pattern Recognition and Image Analysis*, pages 423–431, 2015.

[49] C. Guo and L. Zhang. A simple method for detecting salient regions. *Pattern Recognition*, 42(11):2363–2371, 2009.

[50] C. Guo and L. Zhang. A novel multiresolution spatiotemporal saliency detection model and its applications in image and video compression. *IEEE Trans. Image Process.*, 19(1):185–198, 2010.

[51] J. Harel, C. Koch, and P. Perona. Graph-based visual saliency. In *Proc. NIPS*, pages 545–552, 2006.

[52] M. Havaei, A. Davy, D. Warde-Farley, A. Biard, A. Courville, Y. Bengio, C. Pal, P.-M. Jodoin, and H. Larochelle. Brain tumor segmentation with deep neural networks. *Medical Image Analysis*, 35:18–31, 2017.

[53] K. He and J. Sun. Convolutional neural networks at constrained time cost. In *Proc. CVPR*, pages 5353–5360, 2015.

[54] K. He, X. Zhang, S. Ren, and J. Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. In *Proc. ECCV*, pages 346–361, 2014.

[55] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proc. CVPR*, pages 770–778, 2016.

[56] G. Hinton, N. Srivastava, and K. Swersky. Lecture 6.5-RMSprop: divide the gradient by a running average of its recent magnitude. 2012.

[57] D. Hoiem, Y. Chodpathumwan, and Q. Dai. Diagnosing error in object detectors. *Proc. ECCV*, pages 340–353, 2012.

[58] S. Hua, J. Wua, and L. Xub. Real-time traffic congestion detection based on video analysis. *Journal of Information and Computational Science*, 9(10):2907–2914, 2012.

[59] C. Hua-Tsung, T. Li-Wu, G.Hui-Zhen, L. Suh-Yin, and L. B-SP. Traffic congestion classification for nighttime surveillance videos. In *Proc. ICMEW*, pages 169–174, 2012.

[60] S.-C. Huang and B.-H. Chen. Automatic moving object extraction through a real-world variable-bandwidth network for traffic monitoring systems. *IEEE Trans. Ind. Electron.*, 61(4):2099–2112, 2014.

[61] Y. Huang, R. Wu, Y. Sun, W. Wang, and X. Ding. Vehicle logo recognition system based on convolutional neural networks with a pretraining strategy. *IEEE Trans. Intell. Transp. Syst.*, 16(4):1951–1960, 2015.

[62] F. Iandola, M. Moskewicz, S. Karayev, R. Girshick, T. Darrell, and K. Keutzer. Densenet: Implementing efficient convnet descriptor pyramids. *arXiv preprint arXiv:1404.1869*, 2014.

[63] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[64] H. Jang, H.-J. Yang, D.-S. Jeong, and H. Lee. Object classification using CNN for video traffic detection system. In *21st Korea-Japan Joint Workshop on Frontiers of Comput. Vis.*, pages 1–4, 2015.

[65] J.Cohen. A coefficient of agreement for nominal scales. *Educational and Psychological Measurement*, 20(1):37–46, 1960.

[66] S. Ji, W. Xu, M. Yang, and K. Yu. 3d convolutional neural networks for human action recognition. *IEEE Trans. Pattern Anal. Mach. Intell.*, 35(1):221–231, 2013.

[67] H. Jiang, J. Wang, Z. Yuan, Y. Wu, N. Zheng, and S. Li. Salient object detection: A discriminative regional feature integration approach. In *Proc. CVPR*, pages 2083–2090, 2013.

[68] P. Jiang, H. Ling, J. Yu, and J. Peng. Salient region detection by ufo: Uniqueness, focusness and objectness. In *Proc. ICCV*, pages 1976–1983, 2013.

[69] J. Jin, K. Fu, and C. Zhang. Traffic sign recognition with hinge loss trained convolutional neural networks. *IEEE Trans. Intell. Transp. Syst.*, 15(5):1991–2000, 2014.

[70] H. Jung, M.-K. Choi, J. Jung, J.-H. Lee, S. Kwon, and W. Y. Jung. Resnet-based vehicle classification and localization in traffic surveillance systems. In *Proc. CVPRW*, pages 934–940, 2017.

[71] Y.-K. Jung, K.-W. Lee, and Y.-S. Ho. Content-based event retrieval using semantic scene interpretation for automated traffic surveillance. *IEEE Trans on Int. Trans. Sys*, 2(3):151–163, 2001.

[72] V. Kastrinaki, M. Zervakis, and K. Kostas. A survey of video processing techniques for traffic applications. *Image and vision computing*, 21(4):359–381, 2003.

[73] K.Derpanis and R. Wildes. Classification of traffic video based on a spatiotemporal orientation analysis. In *Workshop on App. of Comp. Vis.*, pages 606–613, 2011.

[74] B. S. Kerner. *Introduction to modern traffic flow theory and control: the long road to three-phase traffic theory*. Springer Science & Business Media, 2009.

[75] P.-K. Kim and K.-T. Lim. Vehicle type classification using bagging and convolutional neural network on multi view surveillance image. In *Proc. CVPRW*, pages 914–919, 2017.

[76] D. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[77] V. Koltun. Efficient inference in fully connected crfs with gaussian edge potentials. In *Proc. NIPS*, pages 109–117, 2011.

[78] P. Krähenbühl and V. Koltun. Geodesic object proposals. In *Proc. ECCV*, 2014.

[79] J. Krause, M. Stark, J. Deng, and L. Fei-Fei. 3d object representations for fine-grained categorization. In *Proc. CVPRW*, pages 554–561, 2013.

[80] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proc. NIPS*, pages 1097–1105, 2012.

[81] S. S. Kruthiventi, V. Gudisa, J. H. Dholakiya, and R. Venkatesh Babu. Saliency unified: A deep architecture for simultaneous eye fixation prediction and salient object segmentation. In *Proc. CVPR*, pages 5781–5790, 2016.

[82] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proc. IEEE*, 86(11):2278–2324, 1998.

[83] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[84] J. Lee and A. Bovik. Estimation and analysis of urban traffic flow. In *IEEE ICIP*, pages 1157–1160, 2009.

[85] J. T. Lee and Y. Chung. Deep learning-based vehicle classification using an ensemble of local expert and global networks. In *Proc. CVPRW*, pages 920–925, 2017.

[86] C. Li, Y. Yuan, W. Cai, Y. Xia, and D. Feng. Robust saliency detection via regularized random walks ranking. In *Proc. CVPR*, pages 2710–2717, 2015.

[87] G. Li and Y. Yu. Visual saliency based on multiscale deep features. In *Proc. CVPR*, pages 5455–5463, 2015.

[88] G. Li and Y. Yu. Deep contrast learning for salient object detection. In *Proc. CVPR*, pages 478–487, 2016.

[89] X. Li, Y. She, D. Luo, and Z. Yu. A traffic state detection tool for freeway video surveillance system. *Procedia-Social and Behavioral Sciences*, 96:2453–2461, 2013.

[90] Y. Li, X. Hou, C. Koch, J. Rehg, and A. Yuille. The secrets of salient object segmentation. In *Proc. CVPR*, pages 280–287, 2014.

[91] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft coco: Common objects in context. In *Proc. ECCV*, pages 740–755, 2014.

[92] F. Liu and M. Gleicher. Region enhanced scale-invariant saliency detection. In *Proc. ICME*, page 1477–1480, 2006.

[93] H. Liu, S. Chen, and N. Kubota. Intelligent video systems and analytics: A survey. *IEEE Trans. Ind. Informat.*, 9(3):1222–1233, 2013.

[94] N. Liu, J. Han, D. Zhang, S. Wen, and T. Liu. Predicting eye fixations using convolutional neural networks. In *Proc. CVPR*, pages 362–370, 2015.

[95] T. Liu, Z. Yuan, J. Sun, J. Wang, N. Zheng, X. Tang, and H. Shum. Learning to detect a salient object. *IEEE Trans. Pattern Anal. Mach. Intell.*, 33(2): 353–367, 2011.

[96] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg. Ssd: Single shot multibox detector. In *Proc. ECCV*, pages 21–37, 2016.

[97] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *Proc. CVPR*, pages 3431–3440, 2015.

[98] W. Luo, X. Zhao, and T.-K. Kim. Multiple object tracking: A review. *arXiv preprint arXiv:1409.7618*, 1, 2014.

[99] Z. Luo, P.-M. Jodoin, S.-Z. Li, and S.-Z. Su. Traffic analysis without motion features. In *Proc. ICIP*, pages 3290–3294, 2015.

[100] Z. Luo, P.-M. Jodoin, S.-Z. Su, S.-Z. Li, and H. Larochelle. Traffic analytics with low frame rate videos. *IEEE Trans. Circuits Syst. Video Technol.*, 2016.

[101] Z. Luo, F. B.-Charron, C. Lemaire, J. Konrad, S. Li, A. Mishra, A. Achkar, J. Eichel, and P.-M. Jodoin. Mio-tcd: A new benchmark dataset for vehicle classification and localization. *Submitted to IEEE Transactions on Image Processing*, 2017.

[102] Z. Luo, J. Eichel, A. Achkar, C. Lemaire, J. Konrad, A. Mishra, S. Li, F. B-Charron, and P.-M. Jodoin. Traffic surveillance workshop and challenge (cvpr 2017). http://tcd.miovision.com, 2017.

[103] Z. Luo, A. Mishra, A. Achkar, J. Eichel, S. Li, and P.-M. Jodoin. Non-local deep features for salient object detection. In *Proc. CVPR*, 2017.

[104] Y. Lv, Y. Duan, W. Kang, Z. Li, and F.-Y. Wang. Traffic flow prediction with big data: a deep learning approach. *IEEE Trans. Intell. Transp. Syst.*, 16(2): 865–873, 2015.

[105] D. Martin, C. Fowlkes, D. Tal, and J. Malik. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In *Proc. ICCV*, volume 2, pages 416–423, 2001.

[106] F. Mehboob, M. Abbas, R. Jiang, M. A. Tahir, S. Al-Maadeed, and A. Bouridane. Automated vehicle density estimation from raw surveillance videos. In *SAI Computing Conference*, pages 1024–1030, 2016.

[107] P. Mehrani and O. Veksler. Saliency segmentation based on learning and graph cut refinement. In *Proc. BMVC*, pages 1–12, 2010.

[108] N. Miller, M. A. Thomas, J. A. Eichel, and A. Mishra. A hidden markov model for vehicle detection and counting. In *Proc. 12th IEEE Conf. Comput. Robot Vis.*, pages 269–276, 2015.

[109] F. Milletari, N. Navab, and S.-A. Ahmadi. V-net: Fully convolutional neural networks for volumetric medical image segmentation. *arXiv preprint:1606.04797*, 2016.

[110] B. T. Morris, C. Tran, G. Scora, M. M. Trivedi, and M. J. Barth. Real-time video-based traffic measurement and visualization system for energy/emissions. *IEEE Trans. Intell. Transp. Syst.*, 13(4):1667–1678, 2012.

[111] D. Mumford and J. Shah. Optimal approximations by piecewise smooth functions and associated variational problems. *Communications on pure and applied mathematics*, 42(5):577–685, 1989.

[112] A. Ng. CS229 lecture notes. http://cs229.stanford.edu/notes/cs229-notes1.pdf.

[113] J. Pan, E. Sayrol, X. G. i Nieto, K. McGuinness, and N. O'Connor. Shallow and deep convolutional networks for saliency prediction. In *Proc. CVPR*, pages 598–606, 2016.

[114] A. Paszke et al. Pytorch. https://github.com/pytorch/pytorch, 2016.

[115] F. Perazzi, P. Krähenbühl, Y. Pritch, and A. Hornung. Saliency filters: Contrast based filtering for salient region detection. In *Proc. CVPR*, pages 733–740, 2012.

[116] F. Porikli and X. Li. Traffic congestion estimation using hmm models without vehicle tracking. In *IEEE Intelligent Vehicles Symposium*, pages 188–193, June 2004.

[117] Y. Qin, H. Lu, Y. Xu, and H. Wang. Saliency detection via cellular automata. In *Proc. CVPR*, pages 110–119, 2015.

[118] J. Redmon and A. Farhadi. Yolo9000: better, faster, stronger. *arXiv preprint arXiv:1612.08242*, 2016.

[119] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. In *Proc. CVPR*, pages 779–788, 2016.

[120] S. Ren, K. He, R. Girshick, and J. Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In *Proc. NIPS*, pages 91–99, 2015.

[121] A. Riaz and S. A. Khan. Traffic congestion classification using motion vector statistical features. In *Sixth International Conference on Machine Vision (ICMV 13)*, volume 9067, page 90671A, 2013.

[122] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[123] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. Imagenet large scale visual recognition challenge. *Int. J. Comput. Vision*.

[124] A. Sharif Razavian, H. Azizpour, J. Sullivan, and S. Carlsson. Cnn features off-the-shelf: an astounding baseline for recognition. In *Proc. CVPRW*, pages 806–813, 2014.

[125] K. Shunsuke, M. Yasuyuki, I. Katsushi, and S. Masao. Traffic monitoring and accident detection at intersections. *IEEE Trans. Intell. Transp. Syst.*, 1(2): 108–118, 2000.

[126] K. Simonyan and A. Zisserman. Two-stream convolutional networks for action recognition in videos. In *Proc. NIPS*, pages 568–576, 2014.

[127] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[128] S. Sivaraman and M. M. Trivedi. Looking at vehicles on the road: A survey of vision-based vehicle detection, tracking, and behavior analysis. *IEEE Trans. Intell. Transp. Syst.*, 14(4):1773–1795, 2013.

[129] A. Sobral, L. Oliveira, L. Schnitman, and F. Souza. Highway traffic congestion classification using holistic properties. In *10th IASTED International Conference on Signal Processing, Pattern Recognition and Applications*, 2013.

[130] P.-L. St-Charles, G.-A. Bilodeau, and R. Bergevin. A self-adjusting approach to change detection based on background word consensus. In *IEEE Winter Conference on Applications of Computer Vision*, pages 990–997, 2015.

[131] P.-L. St-Charles, G.-A. Bilodeau, and R. Bergevin. Subsense: A universal change detection method with local adaptive sensitivity. *IEEE Trans. Image Process.*, 24(1):359–373, 2015.

[132] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision. In *Proc. CVPR*, pages 2818–2826, 2016.

[133] A. A. Taha and A. Hanbury. Metrics for evaluating 3d medical image segmentation: analysis, selection, and tool. *BMC medical imaging*, 15(1):29, 2015.

[134] R. Theagarajan, F. Pala, and B. Bhanu. Eden: Ensemble of deep networks for vehicle classification. In *Proc. CVPRW*, pages 906–913, 2017.

[135] B. Tian, Q. Yao, Y. Gu, K. Wang, and Y. Li. Video processing techniques for traffic flow monitoring: A survey. In *IEEE Conf. Intel. Transp. Syst.*, pages 1103–1108, 2011.

[136] N. Tong, H.Lu, X. Ruan, and M.-H. Yang. Salient object detection via bootstrap learning. In *Proc. CVPR*, pages 1884–1892, 2015.

[137] J. R. Uijlings, K. E. Van De Sande, T. Gevers, and A. W. Smeulders. Selective search for object recognition. *Int. J. Comput. Vision*, 104(2):154–171, 2013.

[138] A. Vedaldi and K. Lenc. Matconvnet: Convolutional neural networks for matlab. In *Proc. ACM Int. Conf. Multimedia*, pages 689–692, 2015.

[139] A. Vedaldi and A. Zisserman. Efficient additive kernels via explicit feature maps. *IEEE Trans. Pattern Anal. Mach. Intell.*, 34(3):480–492, 2012.

[140] A. Vedaldi, K. Lenc, and A. Gupta. Matconvnet: Convolutional neural networks for matlab. http://www.vlfeat.org/matconvnet/matconvnet-manual.pdf.

[141] L. A. Vese and T. F. Chan. A multiphase level set framework for image segmentation using the mumford and shah model. *Int. J. Comput. Vision*, 50(3): 271–293, 2002.

[142] J. Wang, J. Yang, K. Yu, F. Lv, T. Huang, and Y. Gong. Locality-constrained linear coding for image classification. In *Proc. CVPR*, pages 3360–3367, 2010.

[143] L. Wang, H. Lu, X. Ruan, and M. Yang. Deep networks for saliency detection via local estimation and global search. In *Proc. CVPR*, pages 3183–3192, 2015.

[144] P. Wang, J. Wang, G. Zeng, J. Feng, H. Zha, and S. Li. Salient object detection for searched web images via global saliency. In *Proc. CVPR*, pages 3194–3201, 2012.

[145] Q. Wang, W. Zheng, and P. Robinson. Grab: Visual saliency via novel graph model and background priors. In *Proc. CVPR*, pages 535–543, 2016.

[146] R. Wang, F. Bunyak, G. Seetharaman, and K. Palaniappan. Static and moving object detection using flux tensor with split gaussian models. In *Proc. CVPRW*, pages 414–418, 2014.

[147] T. Wang, X. He, S. Su, and Y. Guan. Efficient scene layout aware object detection for traffic surveillance. In *Proc. CVPRW*, pages 53–60, 2017.

[148] Y. Wei, F. Wen, W. Zhu, and J. Sun. Geodesic saliency using background priors. In *Proc. ECCV*, pages 29–42, 2012.

[149] B.-F. Wu, C.-C. Kao, J.-H. Juang, and Y.-S. Huang. A new approach to video-based traffic surveillance using fuzzy hybrid information inference mechanism. *IEEE Trans. Intell. Transp. Syst.*, 14(1):485–491, 2013.

[150] Y. Xie, H. Lu, and M.-H. Yang. Bayesian saliency via low and mid level cues. *IEEE Trans. Image Process.*, 22(5):1689–1698, 2013.

[151] Q. Yan, L. Xu, J. Shi, and J. Jia. Hierarchical saliency detection. In *Proc. CVPR*, pages 1155–1162, 2013.

[152] C. Yang, L. Zhang, H. Lu, X. Ruan, and M. Yang. Saliency detection via graph-based manifold ranking. In *Proc. CVPR*, pages 3166–3173, 2013.

[153] L. Yang, P. Luo, C. C. Loy, and X. Tang. A large-scale car dataset for fine-grained categorization and verification. In *Proc. CVPR*, pages 3973–3981, 2015.

[154] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson. How transferable are features in deep neural networks? In *Proc. NIPS*, pages 3320–3328, 2014.

[155] T. Zhang, S. Liu, C. Xu, and H. Lu. Mining semantic context information for intelligent video surveillance of traffic scenes. *IEEE Trans. Ind. Informat.*, 9 (1):149–160, 2013.

[156] W. Zhang, L. Chen, W. Gong, Z. Li, Q. Lu, and S. Yang. An integrated approach for vehicle detection and type recognition. In *Proc. IEEE UIC-ATC-ScalCom*, pages 798–801, 2015.

[157] R. Zhao, W. Ouyang, H. Li, and X. Wang. Saliency detection by multi-context deep learning. In *Proc. CVPR*, pages 1265–1274, 2015.

[158] J. Zheng, Y. Wang, and W. Zeng. CNN based vehicle counting with virtual coil in traffic surveillance video. In *Proc. IEEE Int. Conf. Multimedia Big Data*, pages 280–281, 2015.

[159] Z. Zhigang, X. Guangyou, Y. Bo, S. Dingji, and L. Xueyin. Visatram: A real-time vision system for automatic traffic monitoring. *Image and Vision Computing*, 18(10):781–794, 2000.

[160] Y. Zhou, H. Nejati, T.-T. Do, N.-M. Cheung, and L. Cheah. Image-based vehicle analysis using deep neural network: A systematic study. In *IEEE Int. Conf. Digital Signal Processing*, pages 276–280, 2016.

[161] S. C. Zhu and A. Yuille. Region competition: Unifying snakes, region growing, and bayes/mdl for multiband image segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 18(9):884–900, 1996.

[162] W. Zhu, S. Liang, Y. Wei, and J. Sun. Saliency optimization from robust background detection. In *Proc. CVPR*, pages 2814–2821, 2014.

[163] Z. Zhu, D. Liang, S. Zhang, X. Huang, B. Li, and S. Hu. Traffic-sign detection and classification in the wild. In *Proc. CVPR*, pages 2110–2118, 2016.

[164] K. H. Zou, S. K. Warfield, A. Bharatha, C. M. Tempany, M. R. Kaus, S. J. Haker, W. M. Wells, F. A. Jolesz, and R. Kikinis. Statistical validation of image segmentation quality based on a spatial overlap index 1: Scientific reports. *Academic radiology*, 11(2):178–189, 2004.